

## Bài 9. Giao diện đồ họa (GUI) với WPF

- **Mục đích, yêu cầu:** Bài này giới thiệu tổng quan về ứng dụng GUI và công nghệ Windows Presentation Foundation (WPF), giới thiệu cách tạo lập và sử dụng các Control cơ bản và nâng cao của cửa sổ xây dựng bằng công nghệ WPF. Sau khi học xong bài này sinh viên có thể tạo được một số ứng dụng cụ thể sử dụng WPF.

- **Hình thức tổ chức dạy học:** Lý thuyết, trực tuyến + trực tiếp + tự học

- **Thời gian:** Lý thuyết(online: 03, offline: 03) Tự học, tự nghiên cứu: 05

- **Nội dung:**

1. Ứng dụng GUI .....	2
1.1. GUI là gì?.....	2
1.2. Những thành phần của GUI.....	2
1.3. Làm thế nào người dùng tương tác với GUI?.....	5
1.4. Một số ví dụ về GUI.....	5
2. Công nghệ WPF.....	7
2.1. Giao diện người dùng hiện đại và những thách thức .....	7
2.2. WPF là gì? .....	8
2.3. Các thành phần của WPF.....	14
2.4. Công cụ phát triển WPF .....	20
2.5. Ứng dụng đầu tiên với WPF – Hello World.....	21
3. Bố trí giao diện trong ứng dụng WPF.....	24
3.1. Giới thiệu chung.....	24
3.2. Các dạng Panel thông dụng .....	25
4. Các điều khiển cơ bản trong WPF .....	31
4.1. Tổng quan về tạo lập các điều khiển với WPF .....	31
4.2. Các điều khiển cơ bản trong WPF .....	33
4.3. Ví dụ xây dựng các control trong WPF.....	41
5. Các điều khiển nâng cao trong WPF.....	45
5.1. Hộp lựa chọn phông chữ (Font Chooser).....	45
5.2. Hộp danh mục ảnh (Image ListBox).....	46
5.3. 3 Hộp mở rộng (Expander).....	48
5.4. Hộp soạn văn bản đa năng (RichTextBox).....	50
Tài liệu tham khảo .....	52

## 1. Ứng dụng GUI

### 1.1. GUI là gì?

GUI, viết tắt của cụm từ Graphical User Interface, tạm dịch là **giao diện đồ họa người dùng**. Đây là thuật ngữ ám chỉ cách giao tiếp của người dùng với các thiết bị máy tính thông qua thao tác với chữ viết hay hình ảnh, thay vì sử dụng các câu lệnh phức tạp.

Nhờ vậy, các tương tác của người dùng trên các thiết bị điện tử ngày một đơn giản hơn, từ đây cũng thúc đẩy sự phát triển của ngành công nghiệp các thiết bị thông minh như điện thoại, [máy tính bảng](#),...

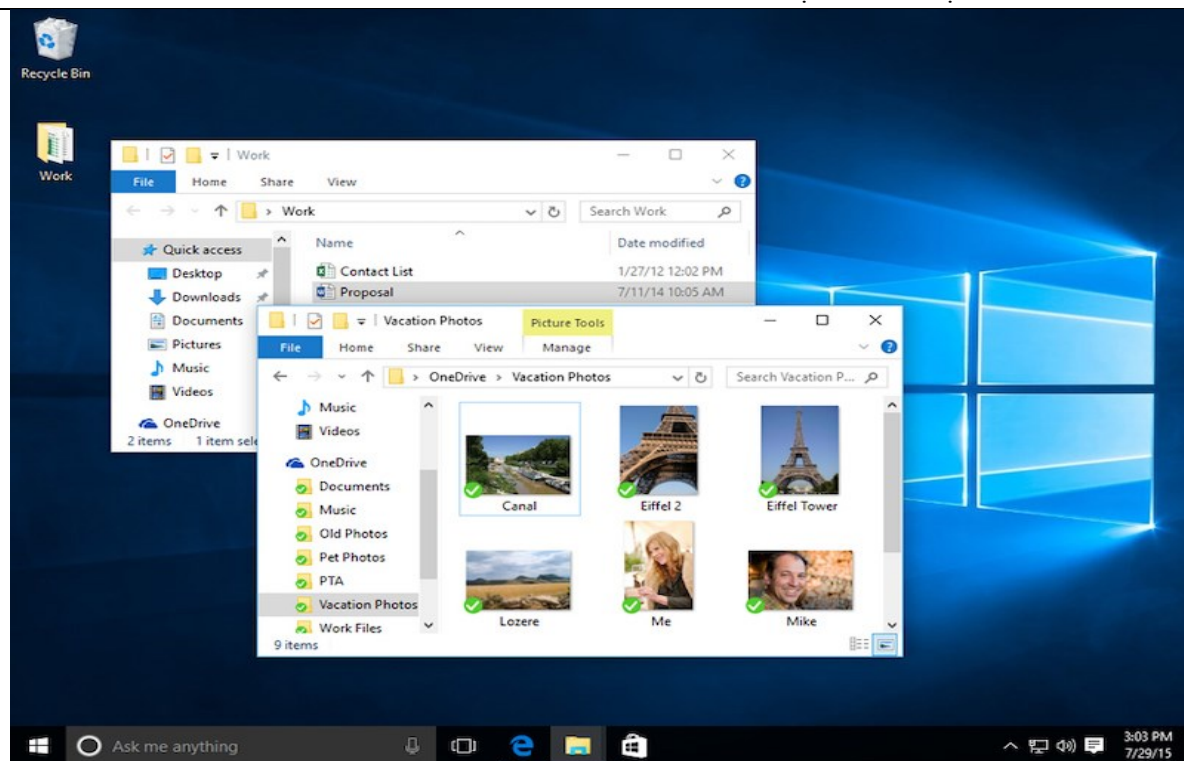


### 1.2. Những thành phần của GUI

#### 1.2.1. Thành phần cấu trúc nên GUI

##### - Cửa sổ làm việc (Windows)

Đây là nơi chứa hết tất cả thông tin mà người dùng có thể tương tác với thiết bị. Thông qua cửa sổ làm việc, người dùng có thể tương tác bằng cách nhấn chọn các biểu tượng, ứng dụng hay kéo thả chúng đến bất kỳ vị trí nào.

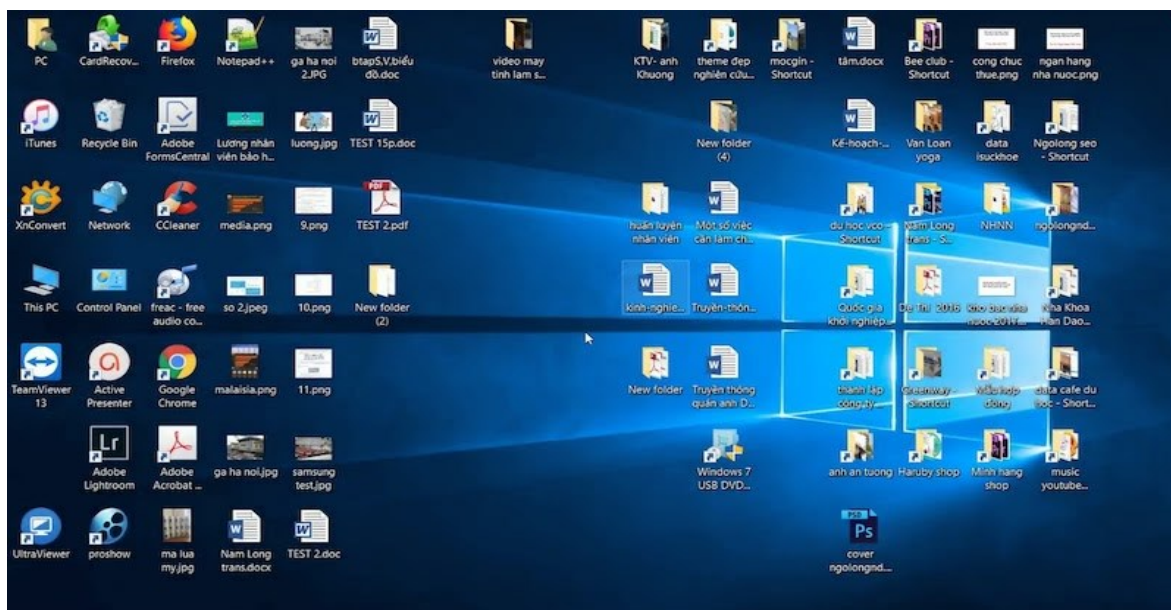


## - Menu

Menu là thành phần thường thấy của GUI, và ở đây người dùng có thể thực hiện lệnh để tương tác với máy tính thông qua danh sách các lựa chọn mà phần mềm cung cấp cho bạn.

## - Biểu tượng (Icon)

Thông thường, các biểu tượng sẽ được hiển thị dưới dạng hình ảnh, giúp cho người dùng có thể tương tác nhanh với máy tính như mở tài liệu, khởi chạy ứng dụng nào đấy. Trong một số trường hợp, người dùng sẽ có thể tìm kiếm tệp thông qua biểu tượng của ứng dụng, từ đó sẽ tiết kiệm thời gian.



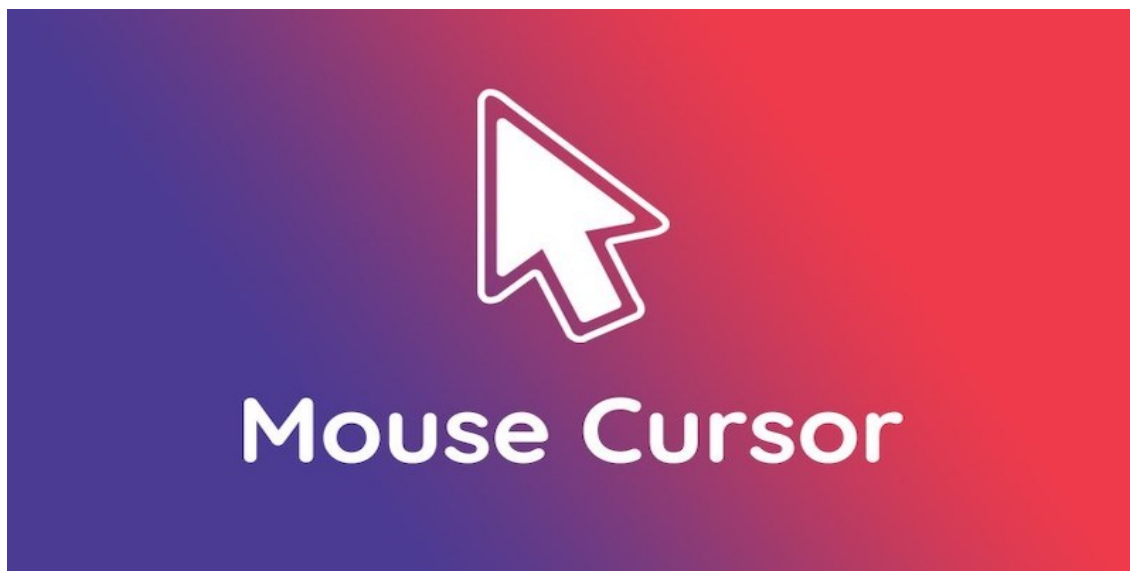
## - Widget

Widget là một thành phần thuộc các ứng dụng, mà ở đây bạn có thể thực hiện một lệnh tương tác cụ thể với ứng dụng đấy.

### 1.2.2. Thành phần tương tác trên GUI

#### - Con trỏ

Đây là thành phần giúp bạn định hướng được vị trí bạn sẽ tương tác, chẳng hạn như vị trí mà con trỏ chuột bạn click hay vị trí mà bạn sẽ nhập liệu các ký tự.

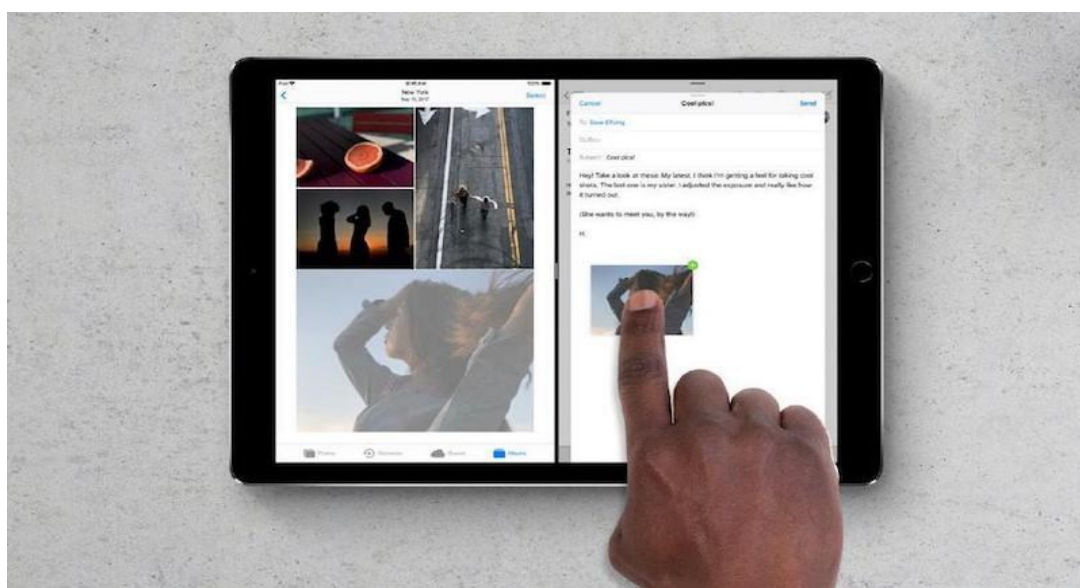


#### - Thao tác chọn

Bạn có thể thực hiện thao tác chọn với các thành phần có trên cửa sổ làm việc. Tương tác chọn có thể được thực hiện bởi cả chuột, bàn phím, bút cảm ứng,...

#### - Thao tác kéo thả

Thông thường, đây là thao tác được sử dụng để tương tác với các tệp hay hình ảnh trong cửa sổ làm việc.





### 1.3. Làm thế nào người dùng tương tác với GUI?

Thông thường, người dùng sẽ sử dụng các thiết bị như **chuột** để điều khiển các thao tác trên thiết bị của mình. Và những năm gần đây thì với sự phát triển của smartphone, các **thao tác cảm ứng** được sử dụng nhiều hơn trong việc tương tác với GUI.

Ngoài ra, một số GUI cũng cho phép người dùng tương tác bằng **bàn phím**, và mặc dù không rườm rà như cách gõ lệnh, cách này cũng ít người sử dụng bởi vì không tiện lợi bằng thao tác cảm ứng hay dùng chuột.



### 1.4. Một số ví dụ về GUI

#### - GNOME Shell

GNOME Shell lần đầu được giới thiệu vào năm 2011 với phiên bản thứ 3, và nó được viết dựa trên nền **ngôn ngữ C** và JavaScript. Giao diện này cũng được áp dụng trên cả máy tính hay điện thoại, và người dùng cũng có thể tương tác thông qua chuột, bàn phím hoặc thao tác cảm ứng.



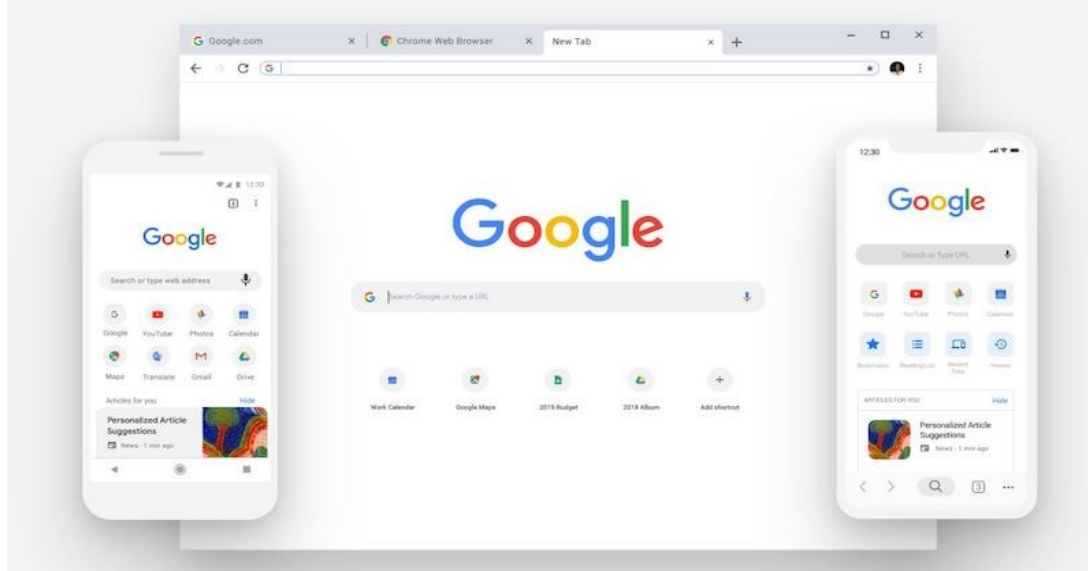
### - Các ứng dụng đến từ Microsoft

Một số phần mềm đến từ bộ [Microsoft Office](#) mà bạn thường thấy như Word, Excel hay Powerpoint cũng được trang bị GUI giúp cho người dùng chúng ta có thể dễ dàng thao tác trong việc nhập liệu dữ liệu, kéo thả hình ảnh,...



## - Trình duyệt Internet

GUI cũng được áp dụng vào các trình duyệt Internet như Google Chrome hay Microsoft Edge, nhờ vậy mà các thao tác lệnh tìm kiếm trên Internet sẽ trở nên nhanh chóng hơn.



## 2. Công nghệ WPF

### 2.1. Giao diện người dùng hiện đại và những thách thức

Trong các ứng dụng hiện đại, giao diện người dùng trực quan chiếm vị trí hết sức quan trọng. Việc trình diễn đúng thông tin, theo đúng cách và vào đúng thời điểm có thể đem lại những giá trị kinh tế xã hội đáng kể. Với những ứng dụng thương mại, chẳng hạn một ứng dụng bán hàng trực tuyến, việc cung cấp một giao diện người dùng mạnh có thể tạo nên sự khác biệt giữa một công ty với các đối thủ cạnh tranh, góp phần làm tăng tăng doanh số và giá trị thương hiệu của hãng này so với hãng khác. Để có được một giao diện người dùng như vậy, việc tích hợp đồ họa, media, văn bản và các thành phần trực quan khác như một thể thống nhất đóng đóng vai trò mấu chốt. Hãy xem xét một ứng dụng cụ thể trong quản lý và theo dõi bệnh nhân của một bệnh viện nào đó. Với sự phát triển của công nghệ đa phương tiện hiện nay, yêu cầu về giao diện người dùng cho hệ thống mới này sẽ bao gồm:

- Hiện thị hình ảnh và text về bệnh nhân.
- Hiện thị và cập nhật hình ảnh 2 chiều cho biết trạng thái của bệnh nhân như nhịp tim, huyết áp.
- Cung cấp hình ảnh chồng lớp 3 chiều về thông tin của người bệnh.
- Trình diễn những đoạn video siêu âm và những chẩn đoán khác, trong đó, cho phép bác sỹ hay y tá thêm vào các ghi chú.
- Cho phép nhân viên bệnh viện đọc và ghi chú trên những tài liệu mô tả về bệnh nhân và tình trạng của người đó.
- Có khả năng hoạt động như một ứng dụng Windows, trong đó, các nhân viên bệnh viện đều được sử dụng đầy đủ các tính năng, đồng thời có thể chạy trên trình duyệt Web có giới hạn về an ninh, cho phép các bác sỹ truy nhập có hạn chế từ xa qua mạng Internet. Với công nghệ từ trước năm 2006, một giao diện như vậy trên Windows đã có thể xây

dựng được, tuy nhiên, sẽ gặp không ít khó khăn bởi một số nguyên nhân chính sau:

- Có rất nhiều công nghệ khác nhau được sử dụng để làm việc với hình ảnh âm thanh và video. Từ đó được những lập trình viên có khả năng sử dụng tốt nhiều công nghệ như vậy không dễ và chi phí cao cho cả quá trình phát triển cũng như bảo trì ứng dụng.
- Thiết kế một giao diện biểu diễn có hiệu quả tất cả những tính năng như vậy cũng là một thách thức. Nó đòi hỏi phải có những người thiết kế giao diện chuyên nghiệp, bởi lập trình viên phần mềm đơn thuần sẽ không có đủ các kỹ năng cần thiết. Điều này lại dẫn tới những khó khăn phát sinh khi người thiết kế và người lập trình làm việc chung.
- Việc cung cấp một giao diện đầy đủ tính năng, hoạt động được như một ứng dụng Windows riêng biệt trên máy desktop, đồng thời có thể được truy nhập thông qua trình duyệt có thể đòi hỏi phải xây dựng hai phiên bản độc lập sử dụng hai công nghệ khác nhau. Ứng dụng Windows trên desktop sử dụng Windows Forms và các công nghệ thuần Windows khác, trong khi ứng dụng trên trình duyệt lại sử dụng HTML và JavaScript. Do đó, cần phải có hai nhóm phát triển với hai phần kỹ năng khác nhau.

*WPF ra đời chính là để xây dựng một nền tảng chung giải quyết những thách thức đã nêu trên.*

## 2.2. WPF là gì?

WPF, viết tắt của *Windows Presentation Foundation*, là hệ thống API mới hỗ trợ việc xây dựng giao diện đồ họa trên nền Windows. Được xem như thế hệ kế tiếp của WinForms, WPF tăng cường khả năng lập trình giao diện của lập trình viên bằng cách cung cấp các API cho phép tận dụng những lợi thế về đa phương tiện hiện đại. Là một bộ phận của .NET Framework 3.0, WPF sẵn có trong Windows Vista và Windows Server 2008. Đồng thời, WPF cũng có thể hoạt động trên nền Windows XP Service Pack 2 hoặc mới hơn, và cả Windows Server 2003.

WPF được xây dựng nhằm vào ba mục tiêu cơ bản:

- 1) Cung cấp một nền tảng thống nhất để xây dựng giao diện người dùng;
- 2) Cho phép người lập trình và người thiết kế giao diện làm việc cùng nhau một cách dễ dàng;
- 3) Cung cấp một công nghệ chung để xây dựng giao diện người dùng trên cả Windows và trình duyệt Web.

### 2.2.1. Nền tảng thống nhất để xây dựng giao diện người dùng

Trước khi WPF ra đời, việc tạo giao diện người dùng theo những yêu cầu mô tả ở ví dụ trên đòi hỏi sử dụng rất nhiều công nghệ khác nhau (xem Bảng 2.1). Để tạo form, các control và các tính năng kinh điển khác của một giao diện đồ họa Windows, thông thường lập trình viên sẽ chọn Windows Forms, một phần của .NET Framework. Nếu cần hiển thị văn bản, Windows Forms có một số tính năng hỗ trợ văn bản trực tiếp hoặc có thể sử dụng Adobe's PDF để hiển thị văn bản có khuôn dạng cố định. Đối với hình ảnh và đồ họa 2 chiều, lập trình viên sẽ dùng GDI+, một mô hình lập trình riêng



biệt có thể truy nhập qua Windows Forms. Để hiển thị video hay phát âm thanh, lập trình viên lại phải sử dụng Windows Media Player, và với đồ họa 3 chiều, anh ta lại phải dùng Direct3D, một thành phần chuẩn khác của Windows. Tóm lại, quá trình phát triển giao diện người dùng theo yêu cầu trở nên phức tạp, đòi hỏi lập trình viên quá nhiều kỹ năng công nghệ.

	Windows Forms	PDF	Windows Forms/ GDI+	Windows Media Player	Direct3D	WPF
Giao diện đồ họa (form và các control)	x					x
On-screen văn bản	x					x
Fixed-format văn bản		x				x
Hình ảnh			x			x
Video và âm thanh				x		x
Đồ họa 2 chiều			x			x
Đồ họa 3 chiều					x	x

*Bảng 9.1 – Thành phần giao diện theo yêu cầu và những công nghệ chuyên biệt cần thiết để tạo chúng.*

WPF là giải pháp hợp nhất nhằm giải quyết tất cả những vấn đề công nghệ nêu trên, hay nói cách khác, WPF cung cấp nhiều tính năng lập trình giao diện trong cùng một công nghệ đơn nhất. Điều này giúp cho quá trình tạo giao diện người dùng trở nên dễ dàng hơn đáng kể. Hình 9.2 cho thấy một giao diện quản lý và theo dõi bệnh nhân có sự kết hợp của hình ảnh, text, đồ họa 2 chiều/3 chiều và nhiều thông tin trực quan khác. Tất cả đều được tạo ra bằng WPF – lập trình viên không cần viết code để sử dụng các công nghệ chuyên biệt như GDI+ hay Direct3D.



Hình 9.1 – Một giao diện người dùng quản lý và theo dõi bệnh nhân sử dụng WPF có thể kết hợp hình ảnh, text, đồ họa 2 chiều/3 chiều và nhiều tính năng trực quan khác

Tuy nhiên, WPF ra đời không có nghĩa là tất cả những công nghệ nêu trên bị thay thế. Windows Forms vẫn có giá trị, thậm chí trong WPF, một số ứng dụng mới vẫn sẽ sử dụng Windows Forms. Windows Media Player vẫn đóng một vai trò công cụ độc lập để chơi nhạc và trình chiếu video. PDF cho văn bản vẫn tiếp tục được sử dụng. Direct3D vẫn là công nghệ quan trọng trong games và các dạng ứng dụng khác (Trong thực tế, bản thân WPF dựa trên Direct3D để thực hiện mọi biểu diễn đồ họa). Việc tạo ra một giao diện người dùng hiện đại không chỉ là việc hợp nhất các công nghệ sẵn có khác nhau. Nó còn thể hiện ở việc tận dụng lợi điểm của card đồ họa hiện đại. Để giải phóng những hạn chế của đồ họa bitmap, WPF dựa hoàn toàn trên đồ họa vector, cho phép hình ảnh tự động thay đổi kích thước để phù hợp với kích thước và độ phân giải của màn hình mà nó được hiển thị.

Bằng việc hợp nhất tất cả các công nghệ cần thiết để tạo ra một giao diện người dùng vào một nền tảng đơn nhất, WPF đơn giản hóa đáng kể công việc của lập trình viên giao diện. Với việc yêu cầu lập trình viên học một môi trường phát triển duy nhất, WPF góp phần làm giảm chi phí cho việc xây dựng và bảo trì ứng dụng. Và bằng việc cho phép tích hợp đa dạng nhiều cách biểu diễn thông tin trên giao diện người dùng, WPF góp phần nâng cao chất lượng, và theo đó là giá trị công việc, của cách thức người dùng tương tác với ứng dụng trên Windows.

### 2.2.2 Khả năng làm việc chung giữa người thiết kế giao diện và lập trình viên

Trong thực tế, việc xây dựng một giao diện người dùng phức hợp như trong ví dụ về ứng dụng quản lý bệnh nhân trên đòi hỏi những kỹ năng ít thấy ở những lập trình viên

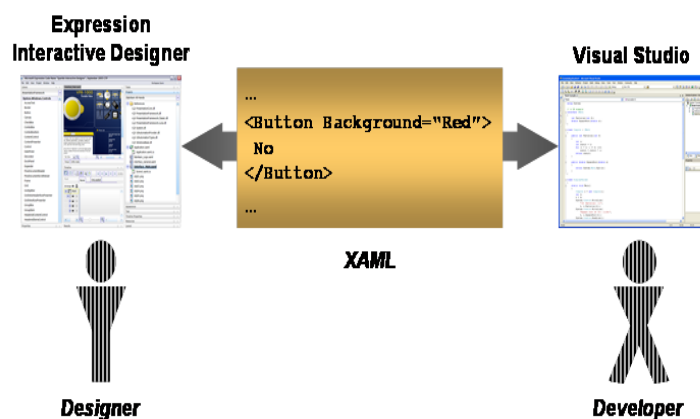
đơn thuần, mà chỉ có thể tìm thấy ở những người thiết kế giao diện chuyên nghiệp. Nhưng câu hỏi đặt ra là làm sao để người thiết kế và lập trình viên có thể làm việc cùng nhau? Thông thường, người thiết kế giao diện sử dụng một công cụ đồ họa để tạo ra những ảnh tĩnh về cách bố trí giao diện trên màn hình. Những hình ảnh này sau đó được chuyển tới lập trình viên với nhiệm vụ tạo ra mã trình để hiện thực hóa giao diện đã thiết kế. Đôi lúc vẽ ra một giao diện thì đơn giản với người thiết kế, nhưng để biến nó thành hiện thực có thể là khó khăn hoặc bất khả thi với lập trình viên. Hạn chế về công nghệ, sức ép tiến độ, thiếu kỹ năng, hiểu nhầm hoặc đơn giản là bất đồng quan điểm có thể khiến lập trình viên không đáp ứng được đầy đủ yêu cầu từ người thiết kế. Do vậy, điều cần thiết ở đây là một cách thức để hai nhóm công tác độc lập này có thể làm việc với nhau mà không làm thay đổi chất lượng của giao diện đã thiết kế. Để thực hiện được điều này, WPF đưa ra ngôn ngữ đặc tả *eXtensible Application Markup Language* (XAML). XAML định ra một tập các phần tử XML như Button, TextBox, Label..., nhằm định nghĩa các đối tượng đồ họa tương ứng như nút bấm, hộp thoại, nhãn..., và nhờ đó cho phép mô tả chính xác diện mạo của giao diện người dùng. Các phần tử XAML cũng chứa các thuộc tính, cho phép thiết lập nhiều tính chất khác nhau của đối tượng đồ họa tương ứng. Ví dụ, đoạn mã sau sẽ tạo ra một nút bấm màu đỏ có nhãn đề “Bỏ qua”.

```
<Button Background="Red">No</Button>
```

Mỗi phần tử XAML lại tương ứng với một lớp WPF, và mỗi thuộc tính của phần tử đó lại tương ứng với thuộc tính hay sự kiện của lớp này. Chẳng hạn, nút bấm màu đỏ trong ví dụ trên có thể tạo bằng C# code như sau:

```
Button btn = new Button();  
btn.Background = Brushes.Red;  
btn.Content = "No";
```

Nếu như mọi thứ có thể biểu diễn bằng XAML thì cũng có thể biểu diễn bằng đoạn mã, thì câu hỏi đặt ra là XAML có ý nghĩa gì? Câu trả lời là việc xây dựng các công cụ sinh và sử dụng các đặc tả bằng XML dễ dàng hơn nhiều so với xây dựng một công cụ tương tự làm việc với đoạn mã. Bởi vậy, XAML mở ra một cách thức tốt hơn để lập trình viên và người thiết kế làm việc với nhau. Hình 2.3 minh họa quá trình này.



Hình 9.2 – XAML hỗ trợ lập trình viên và người thiết kế làm việc chung.

Người thiết kế có thể mô tả giao diện người dùng và tương tác với nó thông qua một công cụ, chẳng hạn như *Microsoft Expression Interactive Designer*. Chỉ tập trung vào việc định ra diện mạo và cảm quan cho giao diện đồ họa WPF, công cụ này sinh các đoạn mô tả giao diện thể hiện qua ngôn ngữ XAML. Lập trình viên sau đó sẽ nhập đoạn mô tả XAML đó vào môi trường lập trình, chẳng hạn như Microsoft Visual Studio. Thay vì lập trình viên phải tái tạo lại giao diện từ đầu dựa trên một ảnh tĩnh mà người thiết kế cung cấp, bản thân các đoạn XAML này sẽ được Microsoft Visual Studio biên dịch để tái tạo thành giao diện đồ họa đúng theo mô tả. Lập trình viên chỉ tập trung vào việc viết mã trình cho giao diện được sinh ra, chẳng hạn như xử lý các sự kiện, theo những chức năng đề ra của ứng dụng.

Việc cho phép người thiết kế và lập trình viên làm việc chung như vậy sẽ hạn chế những lỗi phát sinh khi hiện thực hóa giao diện từ thiết kế. Thêm vào đó, nó còn cho phép hai nhóm công tác này làm việc song song, khiến mỗi bước lặp trong quy trình phát triển phần mềm ngắn đi và việc phản hồi được tốt hơn. Vì cả hai môi trường đều có khả năng hiểu và sử dụng XAML, ứng dụng WPF có thể chuyển qua lại giữa hai môi trường phát triển để sửa đổi hay bổ sung giao diện. Với tất cả những lợi điểm này, vai trò của người thiết kế trong việc xây dựng giao diện được đặt lên hàng đầu.

### 2.2.3. Công nghệ chung cho giao diện trên Windows và trên trình duyệt Web

Trong thời đại bùng nổ của Internet, các ứng dụng Web ngày một phát triển. Việc trang bị giao diện người dùng với đầy đủ tính năng như một ứng dụng desktop sẽ thu hút nhiều người sử dụng, và do đó góp phần làm tăng giá trị doanh nghiệp. Tuy nhiên, như đã nêu trong phần đầu, với những công nghệ truyền thống, để phát triển một giao diện đồ họa vừa hoạt động trên desktop vừa trên trình duyệt Web, đòi hỏi phải sử dụng những công nghệ hoàn toàn khác nhau, giống như việc xây dựng hai giao diện hoàn toàn độc lập. Điều này tạo ra chi phí không cần thiết để phát triển giao diện. WPF là một giải pháp cho vấn đề này. Lập trình viên có thể tạo ra một *ứng dụng trình duyệt XAML* (XBAP) sử dụng WPF chạy trên Internet Explore. Trên thực tế, cùng đoạn code này có thể được dùng để sinh ứng dụng WPF chạy độc lập trên Windows. Hình 9.4 minh họa một ứng dụng dịch vụ tài chính hoạt động như một ứng dụng WPF độc lập. Trong khi đó, hình 9.4 minh họa giao diện của cùng ứng dụng chạy trên Internet Explore dưới dạng XBAP.





Hình 9.3. Ứng dụng WPF độc lập cung cấp dịch vụ tài chính chạy trong cửa sổ riêng.



Hình 9.4. Giao diện của cùng ứng dụng nêu trên dưới dạng một XBAP chạy trên Internet Explore.

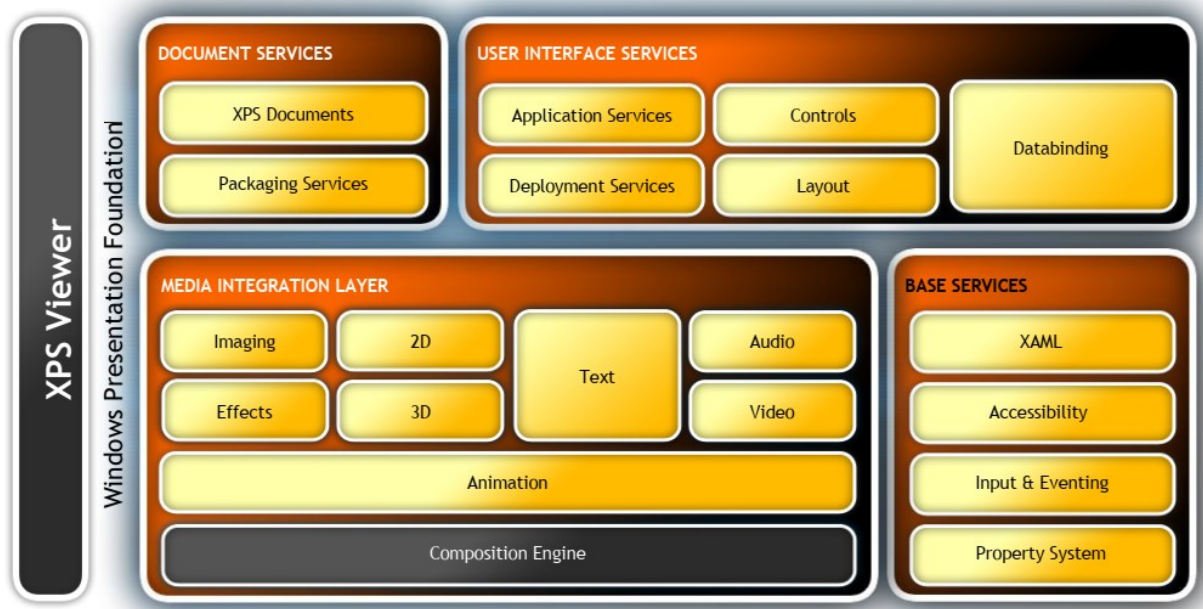
Như đã thấy trong Hình 9.4, phần giao diện của ứng dụng dạng XBAP được trình duyệt chia thành các frame thay vì chạy trên các cửa sổ riêng, ngoài ra, các chức năng đều



được bảo toàn. Cùng một đoạn mã được sử dụng chung cho cả hai trường hợp sẽ làm giảm khối lượng công việc cần thiết để phát triển hai dạng giao diện. Thêm vào đó, sử dụng cùng một đoạn mã cũng có nghĩa là sử dụng cùng kỹ năng của lập trình viên. Do đó, lập trình viên chỉ cần có học một kiến thức chung là có thể sử dụng trong cả hai trường hợp. Một lợi điểm nữa của việc dùng chung công nghệ cho cả giao diện Windows và giao diện Web là người xây dựng ứng dụng không nhất thiết phải quyết định trước loại giao diện nào được sử dụng. Miễn là máy client đáp ứng được những yêu cầu hệ thống để chạy XBAP, một ứng dụng có thể cung cấp cả giao diện Windows và giao diện Web, mà chỉ sử dụng phần lớn những đoạn mã giống nhau. Mỗi ứng dụng XBAP được download khi cần từ một Web server, nên nó phải tuân theo những yêu cầu về an ninh khắt khe hơn đối với một ứng dụng Windows độc lập. Theo đó, XBAP chạy trong phạm vi sandbox an ninh do hệ thống an ninh truy nhập mã của .NET Framework cung cấp. XBAP chỉ chạy với các hệ thống Windows có cài đặt WPF và chỉ với Internet Explore phiên bản 6 và 7 trở lên.

### 2.3. Các thành phần của WPF

Giống như các thành phần khác của .NET Framework, WPF tổ chức các chức năng theo một nhóm *namespace* cùng trực thuộc namespace **System.Windows**. Bất kể chức năng nào được sử dụng, cấu trúc cơ bản của mọi ứng dụng WPF đều gần như nhau. Là ứng dụng Windows độc lập hay là một XBAP, một ứng dụng WPF điển hình bao giờ cũng gồm một tập các trang XAML và phần code tương ứng được viết bằng C# hoặc Visual Basic, còn gọi là các file *code-behind*. Tất cả các ứng dụng đều kế thừa từ lớp chuẩn **Application** của WPF. Lớp này cung cấp những dịch vụ chung cho mọi ứng dụng, chẳng hạn như các biến lưu trữ trạng thái của ứng dụng, các phương thức chuẩn để kích hoạt hay kết thúc ứng dụng. Mặc dù WPF cung cấp một nền tảng thống nhất để tạo giao diện người dùng, những công nghệ mà WPF chứa đựng có thể phân chia thành những thành phần độc lập. Nhân của WPF là cơ chế tạo sinh đồ họa dựa trên vector và độc lập với độ phân giải nhằm tận dụng những lợi thế của phần cứng đồ họa hiện đại. WPF được mở rộng với các tập tính năng phát triển ứng dụng bao gồm XAML, các control, cơ chế móc nối dữ liệu, layout, đồ họa 2 chiều, ba chiều, hoạt họa, style, khuôn dạng mẫu, văn bản, media, text và in ấn. WPF nằm trong .NET Framework, nên ngoài ra, ứng dụng WPF có thể kết hợp các thành phần khác có trong thư viện lớp của .NET Framework.



Hình 9.5. Các thành phần cơ bản của WPF

Phần tiếp theo sẽ giới thiệu sơ lược những thành phần và khái niệm quan trọng của WPF.

### 2.3.1. Layout và Control

Để sắp đặt các thành phần khác nhau trên giao diện, ứng dụng WPF sử dụng *panel*. Mỗi panel có thể chứa các thành phần con, bao gồm các control như nút bấm hay hộp thoại, hay bản thân những panel khác. Những loại panel khác nhau cho phép sắp xếp thành phần con theo những cách khác nhau. Ví dụ, **DockPanel** cho phép các thành phần con có thể được đặt dọc theo cạnh của panel đó, trong khi **Grid** cho phép sắp đặt các thành phần con của nó trên một lưới tọa độ. Giống như bất kỳ một công nghệ giao diện người dùng nào, WPF cung cấp một số lượng lớn các control. Ngoài ra, người dùng có thể tùy ý định nghĩa các control theo ý mình. Các control chuẩn gồm **Button**, **Label**, **TextBox**, **ListBox**, **Menu**, **Slider**, hay phức tạp hơn có **SpellCheck**, **PasswordBox**... Các sự kiện do người dùng tạo ra, như di chuyển chuột hay ấn phím, có thể được các control nắm bắt và xử lý. Trong khi các control và các thành phần giao diện khác có thể được đặc tả đầy đủ bằng XAML, các sự kiện bắt buộc phải được xử lý bằng mã trình.

### 2.3.2. Style và Template

Giống như sử dụng Cascading Style Sheets (CSS) đối với HTML, việc định ra thuộc tính đồ họa cho các đối tượng giao diện một lần, rồi sau đó áp dụng lại cho các đối tượng khác cùng loại thường rất tiện lợi. WPF cũng cung cấp tính năng tương tự bằng việc sử dụng thành phần **Style** của XAML. Ví dụ, kiểu **ButtonStyle** có thể được định nghĩa như sau:

```
<Style x:Key="ButtonStyle">
  <Setter Property="Control.Background" Value="Red"/>
  <Setter Property="Control.FontSize" Value="16"/>
</Style>
```

Bất kỳ nút bấm nào sử dụng kiểu này sẽ có nền màu đỏ và sử dụng font chữ kích thước 16.

Ví dụ:

```
<Button Style="{StaticResource ButtonStyle}">
    Click Here
</Button>
```

Một Style có thể được dẫn xuất từ một Style khác, thừa kế hoặc chồng lên những thuộc tính đã thiết lập. Mỗi style có thể định nghĩa các *trigger* cho phép tạo ra những hiệu ứng tương tác đặc biệt, chẳng hạn như khi lướt chuột qua nút bấm, nút bấm chuyển thành màu vàng.

WPF cũng hỗ trợ sử dụng *template*. Mỗi template tương tự như một style, và ở hai dạng:

- *Template cho dữ liệu*: sử dụng thành phần **DataTemplate** của XAML để thiết lập một nhóm thuộc tính hiển thị của dữ liệu như màu sắc, phương thức căn lề...
- *Template cho control*: sử dụng thành phần **ControlTemplate** của XAML để định ra diện mạo của một control.

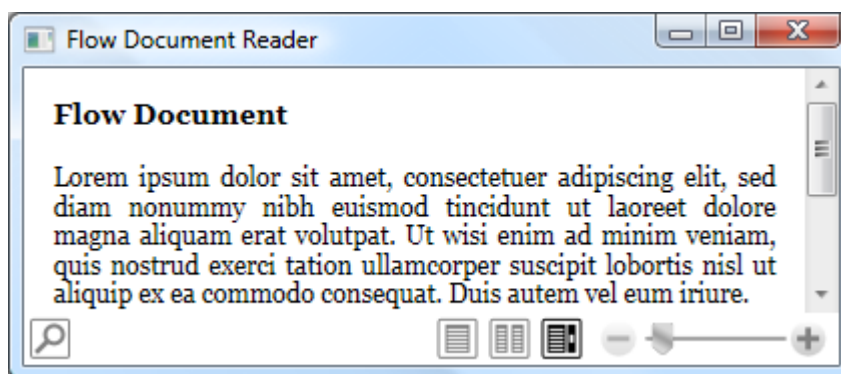
### 2.3.3 Text

Giao diện người dùng ít nhiều đều hiển thị chữ hay text. Đối với phần lớn mọi người, đọc text trên màn hình thường khó hơn đọc trên giấy in. Đó là do chất lượng hiển thị text trên màn hình kém hơn so với khi in ra giấy. WPF tập trung giải quyết vấn đề này, làm chất lượng text hiển thị trên màn hình tương đương trên giấy in. Cụ thể, WPF hỗ trợ các font chữ OpenType chuẩn, cho phép sử dụng các thư viện font đã có. WPF cũng hỗ trợ công nghệ font chữ mới ClearType, cho phép hiển thị các ký tự mịn hơn đối với mắt người, đặc biệt là trên màn hình tinh thể lỏng (LCD). Để nâng cao hơn nữa chất lượng hiển thị text, WPF cho phép một số công nghệ khác như chữ ghép, theo đó một nhóm ký tự được thay thế bằng một ảnh đơn nhất, tạo tâm lý thoải mái hơn khi đọc đối với người dùng.

### 2.3.4 Văn bản

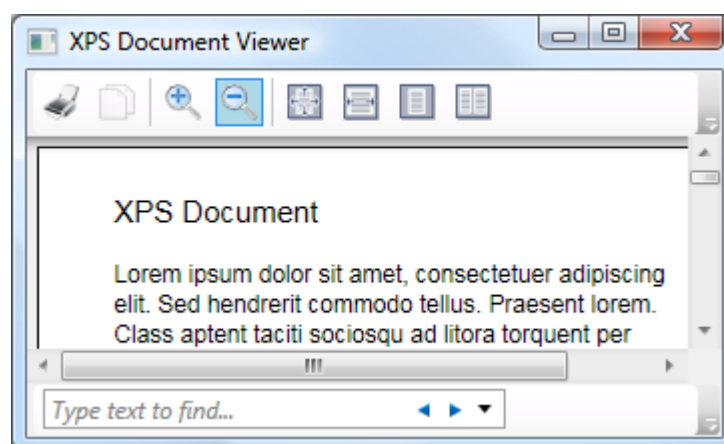
WPF hỗ trợ ba dạng văn bản: văn bản cố định (fixed), văn bản thích nghi (flow/adaptive) và văn bản XPS (XML Paper Specification). Kèm theo đó, WPF cũng cung cấp các dịch vụ để tạo, xem, quản lý, ghi chú, đóng gói và in ấn văn bản. Văn bản cố định trông không đổi bất kể chúng được hiển thị trên màn hình hay in ra máy in. Trong WPF, những văn bản dạng này được định nghĩa bằng phần tử **FixedDocument** trong XAML và được hiển thị bằng control **DocumentViewer**. Trong khi đó, văn bản thích nghi thường chỉ dùng để đọc trên màn hình, và có khả năng tự động thay đổi các thuộc tính hiển thị ảnh và text cho phù hợp với kích thước cửa sổ hay các yếu tố môi trường khác nhằm nâng cao chất lượng đọc cho người dùng. Văn bản thích nghi được định nghĩa bằng phần tử **FlowDocument**. Để hiển thị văn bản thích nghi, WPF sử dụng một số control khác nhau, chẳng hạn như

## FlowDocumentPageViewer, FlowDocumentScrollViewer, FlowDocumentReader...



Hình 9.6. – Một minh họa về văn bản thích nghi trong WPF.

Văn bản XPS xây dựng trên cơ sở văn bản bất động của WPF. XPS là một định dạng mở theo đặc tả XML, có khả năng sử dụng trên nhiều nền tảng khác nhau, được thiết kế nhằm tạo thuận lợi cho việc xây dựng, chia sẻ, in ấn và lưu trữ văn bản. Cũng như văn bản cố định, văn bản XPS được hiển thị bằng **DocumentViewer**.



Hình 9.7. Một minh họa về văn bản XPS trong WPF.

### 2.3.5 Hình ảnh

Trong WPF, hình ảnh được hiển thị nhờ control **Image**, ví dụ:

```
<Image
  Width="200"
  Source="C:\Documents and Settings\All Users\Documents\My Pictures\Ava.jpg" />
```

Control **Image** có thể hiển thị hình ảnh lưu trữ dưới nhiều khuôn dạng khác nhau, bao gồm JPEG, BMP, TIFF, GIF và PNG. Nó cũng có thể hiển thị hình ảnh dạng Windows Media Photo mới được sử dụng trong Windows Vista. Bất kể ở khuôn dạng nào, WPF sử dụng Windows Imaging Component (WIC) để tạo ra hình ảnh. Cùng với các *codec* dùng cho các khuôn dạng ảnh kể trên, WIC cũng cung cấp một nền tảng chung để bổ sung codec khác.

### 2.3.6 Video và Âm thanh

Khi tốc độ của các bộ xử lý và truyền thông mạng ngày một nâng cao, video trở thành

một phần tương tác lớn của người dùng với phần mềm. Người dùng cũng sử dụng nhiều thời gian để nghe nhạc và các dạng âm thanh khác trên máy tính. Do đó, WPF cung cấp tính năng hỗ trợ cả hai dạng media này thông qua phần tử **MediaElement**. Control này có thể chơi các định dạng video WMV, MPEG và AVI, và nhiều định dạng âm thanh khác nhau. Việc lập trình để chạy một đoạn video trở nên khá đơn giản, như trong ví dụ sau:

```
<MediaElement  
Source="C:\Documents and Settings\All Users\Documents\My Videos\Ruby.wmv" />
```

### 2.3.7 Đồ họa hai chiều

Trong 20 năm gần đây, việc tạo ra đồ họa hai chiều trên Windows dựa trên Graphics Device Interface (GDI) và phiên bản sau của nó GDI+. Các ứng dụng Windows Forms phải sử dụng chức năng này thông qua một namespace khác hoàn toàn, bởi bản thân Windows Forms không tích hợp đồ họa 2 chiều. Đối với đồ họa 3 chiều thì càng tồi hơn, Windows Forms phải dựa trên công nghệ hoàn toàn biệt lập là Direct3D. Với WPF, vấn đề trở nên đơn giản hơn nhiều. Cả đồ họa 2 chiều và 3 chiều đều có thể được tạo ra trực tiếp trong XAML hoặc trong code sử dụng thư viện WPF tương ứng. Đối với đồ họa 2 chiều, WPF định ra nhóm control của các khuôn hình (shapes) mà ứng dụng có thể sử dụng để tạo nên hình ảnh, gồm:

- **Line**: vẽ đường thẳng qua 2 điểm.
- **Ellipse**: vẽ ellipse.
- **Rectangle**: vẽ chữ nhật.
- **Polygon**: vẽ đa giác.
- **Polyline**: vẽ đa giác mở.
- **Path**: vẽ hình theo một đường bất kỳ.

Mỗi khuôn hình đều có các thuộc tính phong phú cho phép hiển thị với nhiều tính chất khác nhau: màu nền, màu biên... Một đặc điểm quan trọng trong WPF là: vì mọi thứ đều được xây dựng trên một nền chung, việc kết hợp các đặc tính và đối tượng khác nhau, chẳng hạn, lồng một ảnh vào một hình chữ nhật, trở nên đơn giản. Điểm thú vị nữa là các đối tượng hình học này còn có thể thu nhận các sự kiện từ phía người dùng như một control, chẳng hạn sự kiện nhấp chuột.

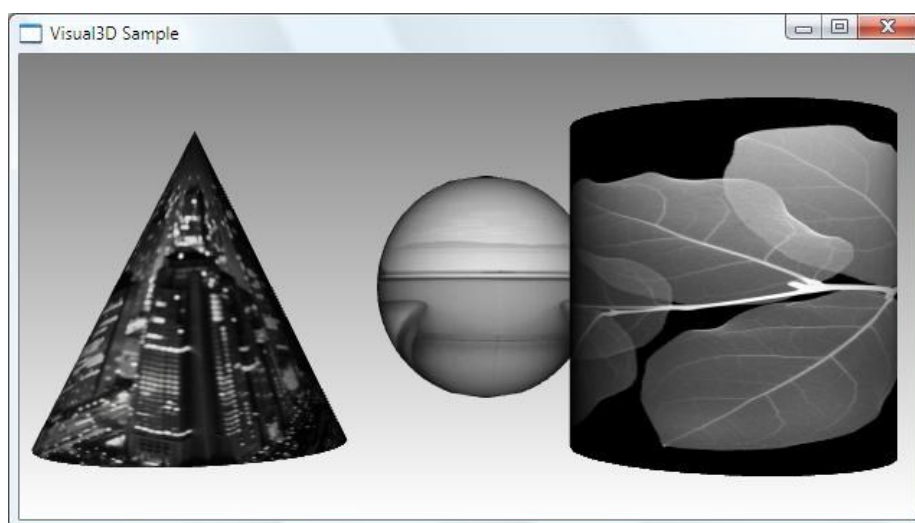
Ngoài ra, WPF cũng cung cấp một nhóm chức năng hình học khác, gọi là *geometries*, để làm việc với đồ họa hai chiều, như **LineGeometry**, **RectangleGeometry**, **EllipseGeometry**, và **PathGeometry**. Dạng hình học này có nhiều thuộc tính và chức năng tương tự như các khuôn hình đã nêu trên. Điểm khác biệt quan trọng nhất là các geometries không được dùng để hiển thị, chúng được dùng chủ yếu để tính toán hình học, ví dụ như để định ra các vùng miền, theo dõi vị trí bấm chuột... Thêm vào đó, WPF cung cấp lớp **Transform** cho phép thực hiện các biến đổi hình học



như xoay, dịch chuyển, co giãn đối tượng đồ họa; hoặc cho phép thực hiện các hiệu ứng hoạt họa theo thời gian thông qua các lớp **Animation** và **Timing**.

### 2.3.8. Đồ họa ba chiều

WPF hỗ trợ đồ họa 3 chiều bằng việc gói các lời gọi API của Direct3D, và do vậy, việc sử dụng chúng trở nên thống nhất và đơn giản hơn đáng kể. Để hiển thị đồ họa ba chiều, ứng dụng WPF sử dụng control **Viewport3D**. Để tạo ra các cảnh ba chiều, lập trình viên mô tả một hay nhiều *mô hình*, sau đó, phân định cách thức các mô hình này được chiếu sáng hay hiển thị. Như thường lệ, điều này được thực hiện bằng XAML, bằng code hay trộn cả hai. Để mô tả mô hình, WPF cung cấp lớp **GeometryModel3D** để tạo ra hình dạng của mô hình. Khi mô hình đã được định hình, diện mạo bên ngoài của nó có thể được điều khiển bằng việc phủ lên các *vật liệu* (material). Chẳng hạn, lớp **SpecularMaterial** cho phép tạo bóng trên bề mặt mô hình. Bất kể được làm từ vật liệu gì, một mô hình có thể được chiếu sáng theo nhiều cách. Lớp **DirectionalLight** cho phép ánh sáng tới từ một hướng xác định, trong khi lớp **AmbientLight** tạo ra ánh sáng đồng đều trên mọi vật trong cảnh. Cuối cùng, để định ra cách nhìn cảnh, lập trình viên phải định ra một *camera*. Ví dụ, **PerspectiveCamera** cho phép phân định khoảng cách từ vị trí nhìn tới vật thể và kiểu nhìn phối cảnh (tuân theo luật gần xa).



Hình 9.8. Tạo lập đối tượng đồ họa ba chiều với WPF.

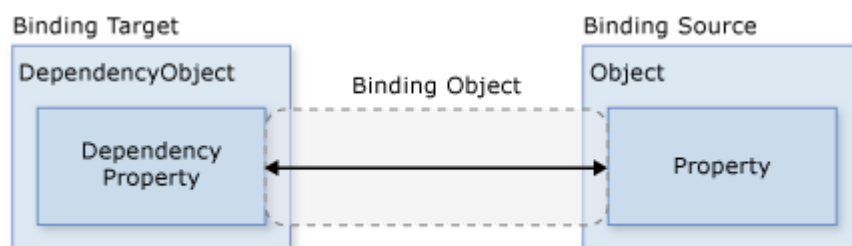
Xây dựng cảnh ba chiều trực tiếp bằng XAML hay mã trình đều không đơn giản. Do đó, chỉ nên dùng ứng dụng WPF để hiển thị cảnh ba chiều, việc xây dựng cảnh nên được thực hiện bằng những công cụ đồ họa chuyên biệt.

### 2.3.9 Móc nối dữ liệu

Phần lớn các ứng dụng được tạo ra đều cung cấp cho người dùng phương tiện để xem và sửa đổi dữ liệu. Trong các ứng dụng WPF, việc lưu trữ và truy xuất dữ liệu đã được thực hiện bởi các công nghệ như Microsoft SQL Server và ADO.NET. Sau khi dữ liệu được truy xuất và tải vào các đối tượng quản lý dữ liệu trên ứng dụng, phần việc khó khăn của ứng dụng WPF mới bắt đầu. Về cơ bản, có hai công việc phải thực hiện:

- 1) Sao chép dữ liệu từ các đối tượng quản lý dữ liệu vào các control trên giao diện, qua đó, dữ liệu có thể được hiển thị hay sửa đổi.
- 2) Đảm bảo rằng những thay đổi trên dữ liệu từ các control được cập nhật trở lại các đối tượng quản lý dữ liệu.

Để đơn giản hóa quá trình phát triển ứng dụng, WPF cung cấp một cơ chế móc nối dữ liệu để thực hiện tự động những bước này. Phần nhân của cơ chế móc nối dữ liệu là lớp **Binding** mà nhiệm vụ của nó là liên kết control trên giao diện (đích) với đối tượng quản lý dữ liệu (nguồn). Mối quan hệ này được minh họa trong hình dưới đây:



Hình 9.9. Quan hệ giữa đối tượng dữ liệu và đối tượng phụ thuộc.

Việc hỗ trợ móc nối dữ liệu được xây dựng ngay từ nhân của WPF. Tất cả các đối tượng đồ họa trong WPF đều kế thừa từ **DependencyObject**, chúng là các *đối tượng phụ thuộc*. Chức năng mà lớp cơ sở này hỗ trợ cho phép thực hiện hiệu ứng hoạt họa, tạo kiểu mẫu (styling) và móc nối dữ liệu. Các đối tượng này đều mang một thuộc tính đặc biệt gọi là **DependencyProperty**, *thuộc tính phụ thuộc*. Phần lớn các thuộc tính hay dùng như **Text**, **Content**, **Width**, **Height**, vân vân đều là các thuộc tính phụ thuộc. Tất cả các thuộc tính phụ thuộc đều có thể tạo hiệu ứng hoạt họa, tạo kiểu và kết nối dữ liệu. Cơ chế móc nối dữ liệu trong WPF còn cung cấp thêm những tính năng như xác thực tính hợp lệ, sắp xếp, lọc và phân nhóm dữ liệu. Thêm vào đó, tính năng móc nối dữ liệu cũng hỗ trợ sử dụng khuôn mẫu dữ liệu (data template) để tạo ra các đối tượng giao diện tùy biến có kết nối dữ liệu, khi các control chuẩn không phù hợp. Móc nối dữ liệu và khuôn dạng dữ liệu có thể được coi là tính năng mạnh nhất của WPF.

## 2.4. Công cụ phát triển WPF

Như đã trình bày ở trên, WPF cung cấp rất nhiều tính năng cho những lập trình viên. Tuy nhiên, một công nghệ dù có hữu dụng đến đâu cũng cần một công cụ và môi trường tốt để phát huy những lợi điểm của nó. Đối với WPF, Microsoft cung cấp một công cụ chuyên dùng cho lập trình viên, và một công cụ khác phục vụ người thiết kế giao diện. Phần dưới đây đề cập ngắn gọn về những công cụ này.

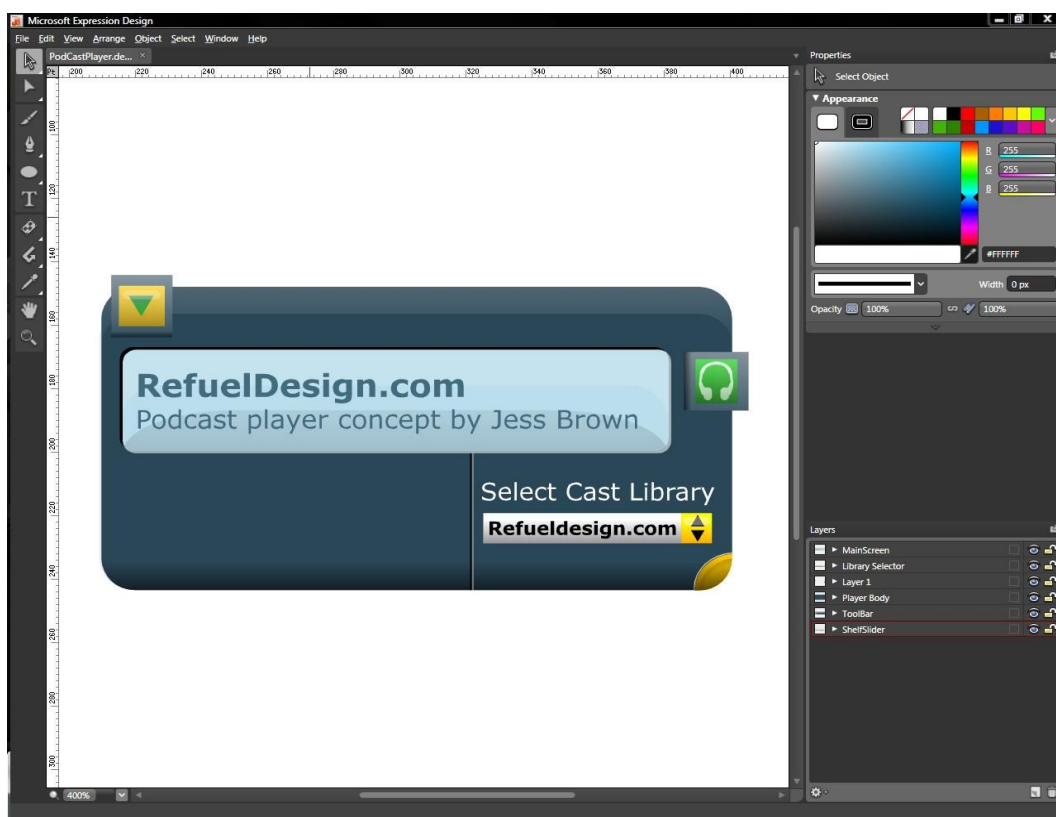
### 2.4.1 Microsoft Visual Studio - Công cụ cho lập trình viên

Visual Studio là công cụ chủ đạo của Microsoft dành cho lập trình viên phần mềm. Microsoft cung cấp thành phần mở rộng cho Visual Studio 2005 cho phép lập trình viên có thể tạo ra các ứng dụng WPF. Phiên bản tiếp theo của Visual Studio (2008) có bổ sung thêm các tính năng phát triển ứng dụng WPF, trong đó bao gồm Visual Designer,

môi trường thiết kế giao diện cho WPF. Sử dụng công cụ này, lập trình viên có thể tạo ra giao diện WPF một cách trực quan, trong khi sản sinh các đặc tả XAML tương ứng một cách tự động.

### 2.4.2 Microsoft Expression Design – Công cụ cho người thiết kế

Như đã giới thiệu trong phần trước, mục tiêu cơ bản của WPF là nâng cao vị thế của người thiết kế trong việc tạo giao diện người dùng. Để đạt mục tiêu này, ngoài XAML là công nghệ cốt lõi, Microsoft cũng đưa ra một công cụ mới cho phép người thiết kế làm việc thuận tiện hơn, đó là *Microsoft Expression Design* (Hình 0.10).



Hình 9.10. Giao diện của công cụ Microsoft Expression Design.

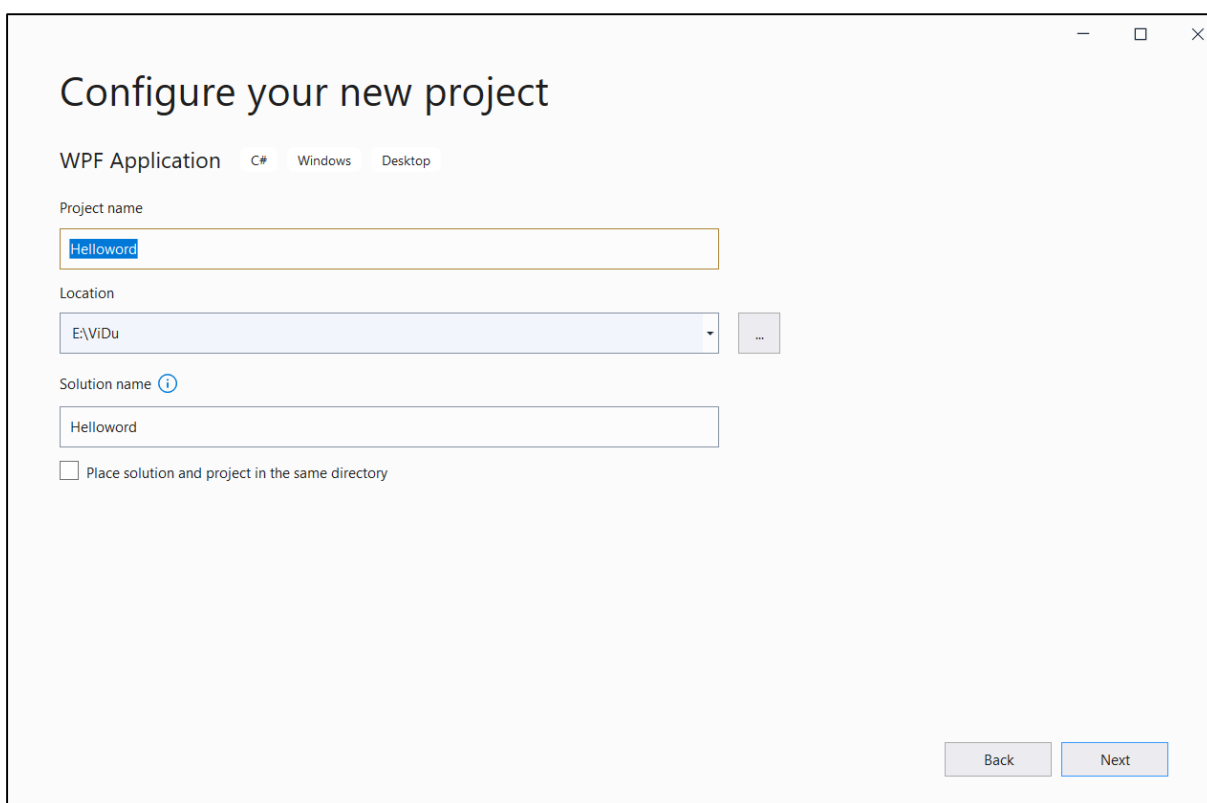
Microsoft Expression Design cung cấp những tính năng truyền thống của một công cụ thiết kế, cho phép người dùng làm việc theo cách quen thuộc. Ngoài ra, công cụ này đặc biệt tập trung vào việc hỗ trợ tạo giao diện cho các ứng dụng WPF. Tất cả các tính năng WPF mô tả ở trên đều sẵn có trong môi trường thiết kế này, và cho phép người dùng thiết kế một cách trực quan. Kết quả thiết kế được biểu diễn dưới dạng file XAML do công cụ này sinh ra, và sau đó có thể được nhập vào môi trường Visual Studio.

### 2.5. Ứng dụng đầu tiên với WPF – Hello World

Phần này giúp các bạn làm quen với lập trình WPF thông qua một ví dụ kinh điển: Hello World. Ứng dụng chỉ bao gồm một nút bấm có nhãn ban đầu là Hello World. Khi nhấp chuột vào nút, nút sẽ đổi tên thành “From Hanoi, Vietnam”. Môi trường lập trình ở đây là bộ Visual Studio 2019 với .NET Framework 5.0

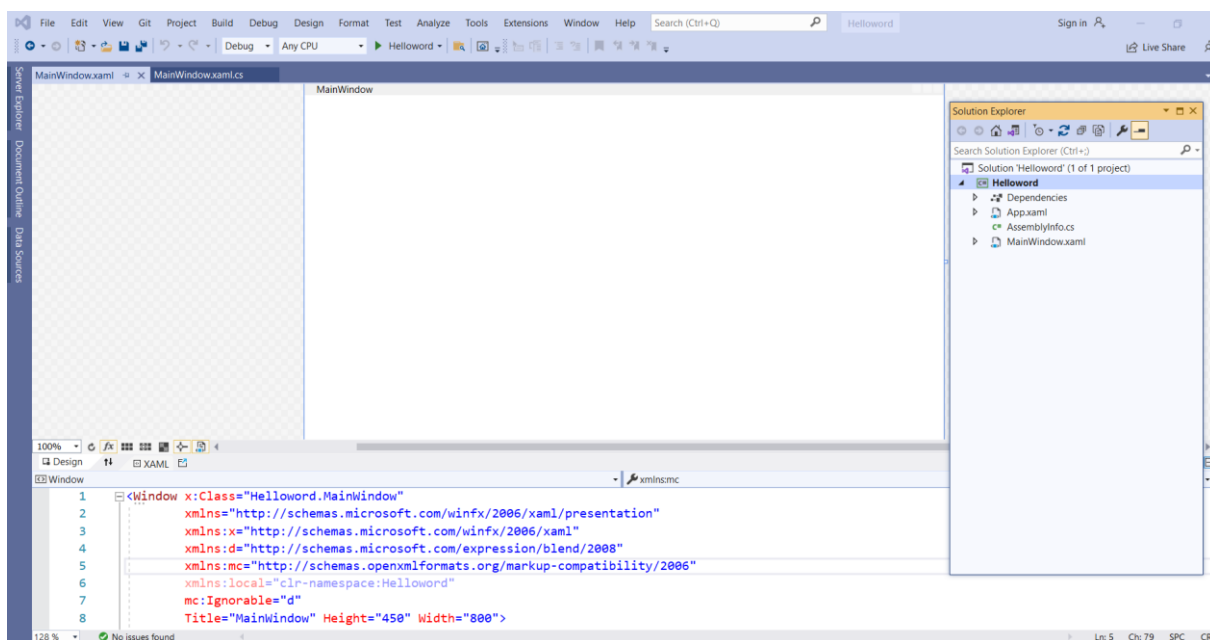
Quy trình thực hiện như sau:

## 2.5.1. Tạo ứng dụng WPF



Ở đây, ta chú ý chọn .Net Framework 5.0.

Giao diện thiết kế ứng dụng:



Chúng ta sẽ làm 2 phương pháp, một là viết code C# trực tiếp trong ứng dụng, hai là viết bằng mã XAML.

## 2.5.2 Tạo ứng dụng Hello World bằng code C#

Khai báo Button trong lớp Window1. Lớp Button được tạo ra từ namespace:

```
System.Windows.Controls;
```

```
namespace Helloworld
{
    public partial class Window1:Window
    {
        // khai báo 1 button
        Button button;
        public Window1()
        {
            InitializeComponent();
        }
    }
}
```

Ở phương thức khởi tạo, ta sẽ lần lượt đặt thuộc tính cho nút bấm:

```
//tạo mới button
button = new Button();
//xác định thuộc tính cho button
button.Content = "Hello World";
button.LayoutTransform = new ScaleTransform(3, 3);
button.Margin = new System.Windows.Thickness(10);
//thêm phương thức xử lý sự kiện Click cho button
button.Click += new RoutedEventHandler(button_Click);
//đưa button vào Window
this.Content = button;
```

Để tạo phương thức xử lý sự kiện Click cho button, chúng ta chỉ thêm lệnh `button.Click +=` và nhấn Tab 2 lần. Code tự sinh ra như sau:

```
void button_Click(object sender, RoutedEventArgs e)
{
    // xử lý button khi người dùng click
}
```

Trong phương thức này, ta thêm vào dòng lệnh:

```
button.Content = "From Hanoi, Vietnam";//đổi nội dung (caption) của button
```

Nhấn F5 để biên dịch project và chạy kết quả.

### 2.5.3 Tạo ứng dụng Hello World bằng XAML

WPF hỗ trợ trên nền tảng XAML nên ta có thể tạo đối tượng hoàn toàn bằng ngôn ngữ XAML. Lưu ý, với cùng project trên, muốn viết đặc tả bằng XAML tương đương ta cần xóa bỏ phần mã trình C# cũ đi, vì C# và XAML không thể cùng sinh một đối tượng. Cách thực hiện:

Mở file `Window1.xaml` tương ứng với file code `Window1.xaml.cs` ở trên. Thêm đoạn mã đặc tả XAML tương đương sau:

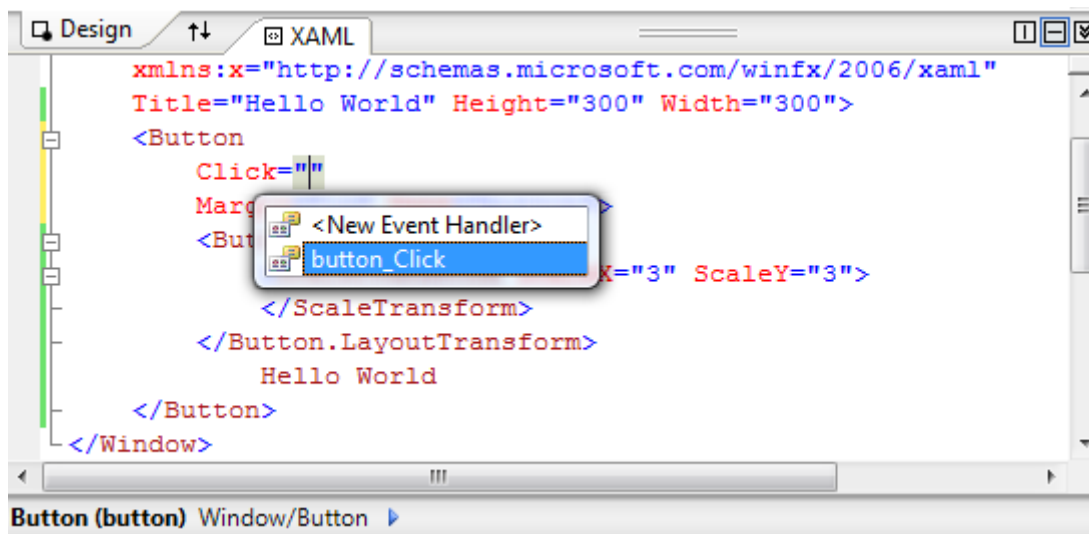
```
<Window x:Class="Helloworld.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window1" Height="300" Width="300">
    <Button Name="button" Margin="10">
        <Button.LayoutTransform>
            <ScaleTransform ScaleX="3" ScaleY="3">
        </ScaleTransform>
    </Button.LayoutTransform>
```



```

Hello World
</Button>
</Window>
    
```

Giờ ta gán tên phương thức xử lý sự kiện *Click* cho nút bấm này:



F5 để chạy ứng dụng. Ta sẽ thu được kết quả tương tự như phần 2.5.2.

### 3. Bố trí giao diện trong ứng dụng WPF

#### 3.1. Giới thiệu chung

Như đã giới thiệu trong bài mở đầu, WPF sử dụng các dạng *panel* khác nhau để bố trí các phần tử trên giao diện người dùng. Điều này xuất phát từ ý tưởng kết hợp công nghệ giao diện mạnh như Windows Forms, với các kỹ thuật sắp đặt (layout) của trình duyệt nhằm nâng cao tính linh hoạt trong việc bố trí các phần tử trên giao diện. Các công nghệ xây dựng giao diện như VB6 form, Access forms... dựa trên nguyên tắc bố trí theo *vị trí tuyệt đối*. Nghĩa là, người lập trình phải xác định giá trị tọa độ góc trên bên trái của một control (so với với góc trên bên trái của một form) khi muốn đặt nó lên form. Điều này cho phép lập trình viên điều khiển vị trí của control khá dễ dàng, nhưng lại thường đòi hỏi một lượng lớn mã trình khi cần thay đổi kích thước form. Đây là phương pháp tiếp cận theo hướng *áp đặt* (imperative), trong đó máy tính được chỉ rõ phải làm những bước gì, khi nào và theo trình tự nào. Với cách thức bố trí này, các điều khiển như Label hay Panel không tự động kéo giãn để phù hợp với kích thước phần nội dung chứa trong nó. Và như vậy, nếu phần nội dung của một Label lớn hơn vùng có thể hiển thị của Label đó, thì nội dung này sẽ bị cắt đi hoặc bị che lấp. Trong khi đó, các phần tử giao diện Web trên trình duyệt được sắp xếp theo phương thức *khai báo* (declarative), trong đó, người lập trình chỉ đưa ra những thứ cần làm, còn máy tính sẽ giải quyết vấn đề làm như thế nào. Với phương thức này, giao diện trên trình duyệt không đòi hỏi mã trình để thay đổi kích thước các *vùng chứa* (container). HTML cho phép ta định ra một chuỗi các vùng chứa, ví dụ như các phần tử `<div>`, `<table>`, `<tr>` và `<td>`, để bố trí các phần tử UI khác trong đó một cách linh động bên phải hoặc bên trái một đối tượng; hay cũng có thể sắp xếp chúng theo vị trí

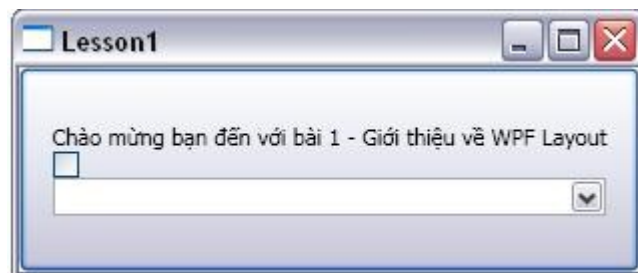
tuyệt đối trên trang Web. Các phần tử như `<div>` quan tâm tới kích thước bên trong nội dung của nó và sẽ tự động giãn ra để chứa đủ nội dung bên trong. Tuy nhiên, cả hai cách tiếp cận nêu trên đều khó có thể đạt được cách bố trí như ý, mặc dù cách bố trí trên trình duyệt có giảm lượng code xử lý. Hiện nay, Windows Forms đưa thêm những khái niệm như *Docking* (cấp bến) hay *Anchoring* (buông neo), bổ sung một cách tiếp cận kiểu khai báo linh hoạt hơn để phát triển các ứng dụng trên máy trạm. WPF tiếp bước xu hướng này với việc bố trí giao diện dựa trên khái niệm về *panel*. Phần lớn các phần tử UI trong ứng dụng WPF chỉ có thể chứa duy nhất một phần tử con. Chẳng hạn, đoạn mã XAML sau sẽ mắc lỗi biên dịch sau: “*The 'Button' object already has a child and cannot add 'CheckBox'. 'Button' can accept only one child.*” Nghĩa là, đối tượng nút bấm ‘Button’ đã chứa một phần tử con (cụ thể là đối tượng ‘TextBlock’) và do đó, không thể thêm vào một đối tượng ‘CheckBox’ hay ‘ComboBox’ nữa. *Đoạn mã XAML sau đây không biên dịch được*

```
<Button>
<TextBlock> Chào mừng bạn đến với bài 1 - Giới thiệu về WPF Layout
</TextBlock>
<CheckBox />
<ComboBox />
</Button>
```

Để nút bấm này có thể chứa 3 phần tử con bên trong nó, WPF sử dụng *panel*. Có nhiều dạng *panel* khác nhau trong WPF và mỗi dạng cho phép một kiểu bố trí giao diện khác nhau. Các *panel* có thể lồng vào nhau cho phép bố trí các phần tử trên giao diện ở những dạng sắp xếp bất kỳ. Ví dụ, để sửa vấn đề nêu ra ở ví dụ trên, ta có thể lồng một *StackPanel* bên trong nút bấm, và bố trí các phần tử con bên trong *panel* đó. *Đoạn mã XAML sau đây cho phép 1 nút bấm chứa được text, checkbox và combobox*

```
<Button>
<StackPanel>
<TextBlock>Chào mừng bạn đến với bài 1 - Giới thiệu về WPF Layout
</TextBlock>
<CheckBox />
<ComboBox />
</StackPanel>
</Button>
```

Kết quả biên dịch sẽ là:



Hình 9.11 – Kết quả sửa đổi đoạn mã XAML hiển thị hơn một phần tử giao diện con trong một nút bấm sử dụng *StackPanel*

Các dạng *Panel* thông dụng

Để bạn đọc thấy được vai trò quan trọng của panel trong việc bố trí giao diện, phần sau đây sẽ lần lượt giới thiệu những dạng panel thường dùng và các đặc tính của chúng thông qua các ví dụ đơn giản.

### 3.2.1. StackPanel

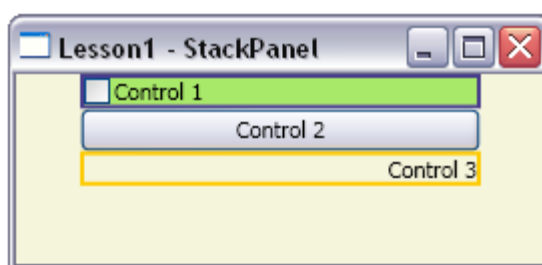
**StackPanel** bố trí các phần tử con nằm trong nó bằng cách sắp xếp chúng theo thứ tự trước sau. Các phần tử sẽ xuất hiện theo thứ tự mà chúng được khai báo trong file XAML theo chiều dọc (ngầm định) hoặc theo chiều ngang.

#### 3.2.1.1 Sắp xếp theo chiều dọc

Sau đây là đoạn mã XAML minh họa việc sắp xếp các phần tử UI trong một đối tượng Window bằng StackPanel theo chiều dọc:

```
<!--Khai báo khởi tạo cửa sổ-->
<Window x:Class="Lesson1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Lesson1 - StackPanel" Height="120" Width="280">
<!--Sử dụng StackPanel sắp xếp ngầm định theo chiều dọc-->
<StackPanel Background="Beige">
<!--Thiết lập thuộc tính khung viền bao quanh control 1-->
<Border Width="200" BorderBrush="DarkSlateBlue" Background="#a9e969"
BorderThickness="2">
<!--Khai báo control 1 dạng checkbox-->
<CheckBox>Control 1</CheckBox>
</Border>
<!--Khai báo control 2 dạng nút bấm-->
<Button Width="200">Control 2</Button>
<!--Thiết lập thuộc tính khung viền bao quanh control 3-->
<Border Width="200" BorderBrush="#feca00" BorderThickness="2">
<!--Khai báo control 3 dạng text-->
<TextBlock HorizontalAlignment="Right">Control 3</TextBlock>
</Border>
</StackPanel>
</Window>
```

Kết quả là:



*Hình 9.12 – Sắp xếp nhiều control theo thứ tự kế tiếp trên xuống dưới sử dụng StackPanel*

Trong trường hợp sắp xếp theo chiều dọc, nếu tổng chiều cao của các phần tử con lớn hơn chiều cao của form chứa, thì các phần tử nằm ngoài form sẽ không được nhìn thấy.

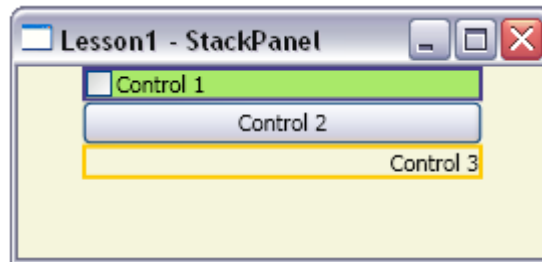
#### 3.2.1.2 Sắp xếp theo chiều ngang

Sau đây là đoạn mã XAML minh họa việc sử dụng StackPanel để sắp xếp các phần tử

UI cùng ví dụ ở trên theo chiều ngang. Điểm khác biệt duy nhất ở đây là thiết lập thêm thuộc tính `Orientation="Horizontal"` của đối tượng `StackPanel` được sử dụng.

```
<!--Sử dụng StackPanel sắp xếp theo chiều ngang xác định bằng thuộc tính
Orientation="Horizontal"-->
<StackPanel Background="Beige" Orientation="Horizontal">
<!--Thiết lập thuộc tính khung viền bao quanh control 1-->
<Border Width="90" Height="80" BorderBrush="DarkSlateBlue" Background="#a9e969"
BorderThickness="2">
<!--Khai báo control 1 dạng checkbox-->
<CheckBox>Control 1</CheckBox>
</Border>
<!--Khai báo control 2 dạng nút bấm-->
<Button Width="90" >Control 2</Button>
<!--Thiết lập thuộc tính khung viền bao quanh control 3-->
<Border Width="90" BorderBrush="#feca00" BorderThickness="2">
<!--Khai báo control 3 dạng text-->
<TextBlock HorizontalAlignment="Right">Control 3</TextBlock>
</Border>
</StackPanel>
```

Và kết quả sẽ là:



*Hình 9.13 – Sắp xếp nhiều control theo thứ tự kế tiếp trái sang phải sử dụng StackPanel*

Trong trường hợp sắp xếp theo chiều ngang, nếu tổng chiều rộng của các phần tử con lớn hơn chiều rộng của form chứa, thì các phần tử nằm ngoài form sẽ không được nhìn thấy.

### 3.2.2. WrapPanel

**WrapPanel** cho phép sắp xếp các phần tử từ trái sang phải. Khi một dòng phần tử đã điền đầy khoảng không gian cho phép theo chiều ngang, `WrapPanel` sẽ cuộn phần tử tiếp theo xuống đầu dòng tiếp theo (tương tự như việc cuộn text). Dưới đây là một ví dụ đơn giản về việc sử dụng `WrapPanel`:

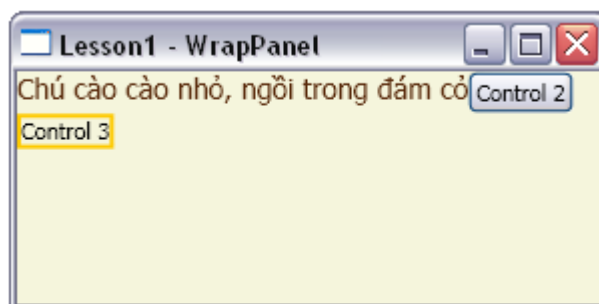
```
<Window x:Class="Lesson1.Window3"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Lesson1 - WrapPanel" Height="150" Width="300">
<!--Sử dụng WrapPanel-->
<WrapPanel Background="Beige">
<TextBlock FontSize="14"
Foreground="#58290A">
Chú cào cào nhỏ, ngồi trong đám cỏ
</TextBlock>
<Button>Control 2</Button>
<Border BorderBrush="#feca00" BorderThickness="2">
```

```

<TextBlock>Control 3</TextBlock>
</Border>
</WrapPanel>
</Window>

```

Do chiều dài tổng cộng của 3 control lớn hơn chiều dài của Window, đồng thời, chiều dài của 2 control đầu (TextBlock và Button) nhỏ hơn chiều dài Window, WrapPanel sẽ xếp TextBlock cuối cùng xuống hàng dưới. Kết quả là:



Hình 9.14 – Sử dụng WrapPanel

### 3.2.3. DockPanel

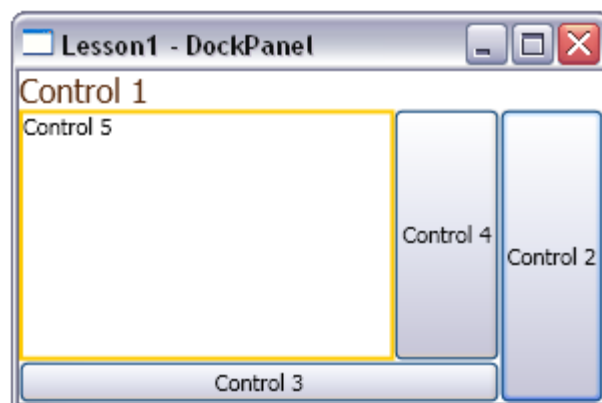
DockPanel cho phép các phần tử bám lên các cạnh của panel DockPanel bao chứa chúng, tương tự như khái niệm Docking trong Windows Forms. Nếu như có nhiều phần tử cùng bám về một cạnh, chúng sẽ tuân theo thứ tự mà chúng được khai báo trong file XAML. Sau đây là đoạn mã XAML minh họa việc sử dụng DockPanel:

```

<DockPanel>
<TextBlock FontSize="16" DockPanel.Dock="Top"
Foreground="#58290A">
Control 1
</TextBlock>
<Button DockPanel.Dock="Right">Control 2</Button>
<Button DockPanel.Dock="Bottom">Control 3</Button>
<Button DockPanel.Dock="Right">Control 4</Button>
<Border BorderBrush="#feca00" BorderThickness="2">
<TextBlock>Control 5</TextBlock>
</Border>
</DockPanel>

```

Phần tử Border cuối cùng sẽ điền vào phần không gian còn lại vì thuộc tính DockPanel.Dock không xác định. Kết quả là:



Hình 9.15 – Sử dụng DockPanel – Control 1 bám vào cạnh trên DockPanel,



*Control 2 và 4 nằm bên phải, Control 3 nằm dưới đáy; Control 5 chiếm toàn bộ không gian còn lại.*

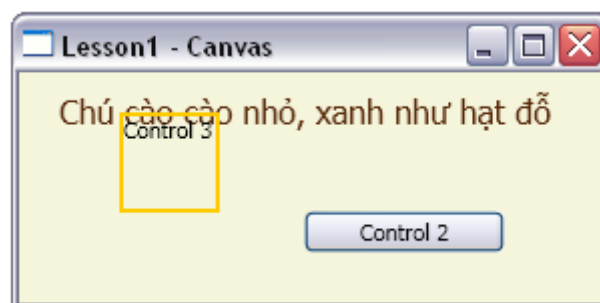
### 3.2.4. Canvas

Panel dạng **Canvas** sử dụng phương thức sắp xếp các phần tử UI theo vị trí tuyệt đối bằng cách đặt thuộc tính Top (đỉnh) và Left (bên trái) của chúng. Thêm vào đó, thay vì đặt thuộc tính Top, Left, ta có thể đặt thuộc tính Bottom (đáy), Right (bên phải). Nếu ta đặt đồng thời thuộc tính Left và Right, thuộc tính Right sẽ bị bỏ qua. Phần tử UI sẽ không thay đổi kích thước để thỏa mãn 2 thuộc tính trên cùng một lúc. Tương tự thuộc tính Top sẽ được ưu tiên hơn thuộc tính Bottom. Các phần tử được khai báo sớm hơn trong file XAML sẽ có thể bị che khuất phía dưới các phần tử được khai báo muộn hơn nếu vị trí của chúng xếp chồng lên nhau.

Sau đây là một ví dụ minh họa việc sử dụng Canvas để sắp xếp các phần tử UI.

```
<!--Sử dụng Canvas-->
<Canvas Background="Beige">
  <TextBlock FontSize="16" Canvas.Top="10" Canvas.Left="20"
    Foreground="#58290A">
    Chú cào cào nhỏ, xanh như hạt đỗ
  </TextBlock>
  <Button Canvas.Bottom="25" Canvas.Right="50" Width="100">Control 2</Button>
  <Border BorderBrush="#feca00" BorderThickness="2" Height="50" Width="50"
    Canvas.Top="20" Canvas.Left="50">
    <TextBlock>Control 3</TextBlock>
  </Border>
</Canvas>
```

Vị trí của phần tử TextBlock đầu tiên và phần tử Border được đặt theo thuộc tính Top, Left, trong khi đó, phần tử Button được sắp vị trí theo thuộc tính Bottom, Right. Border sẽ nằm chồng lên TextBlock đầu tiên vì có sự xếp chồng về vị trí của hai phần tử này. Thêm vào đó, TextBlock đầu được khai báo trước Border trong đoạn mã XAML. Kết quả là:



Hình 9.16 – Sử dụng Canvas để sắp xếp các phần tử UI

### 3.2.5

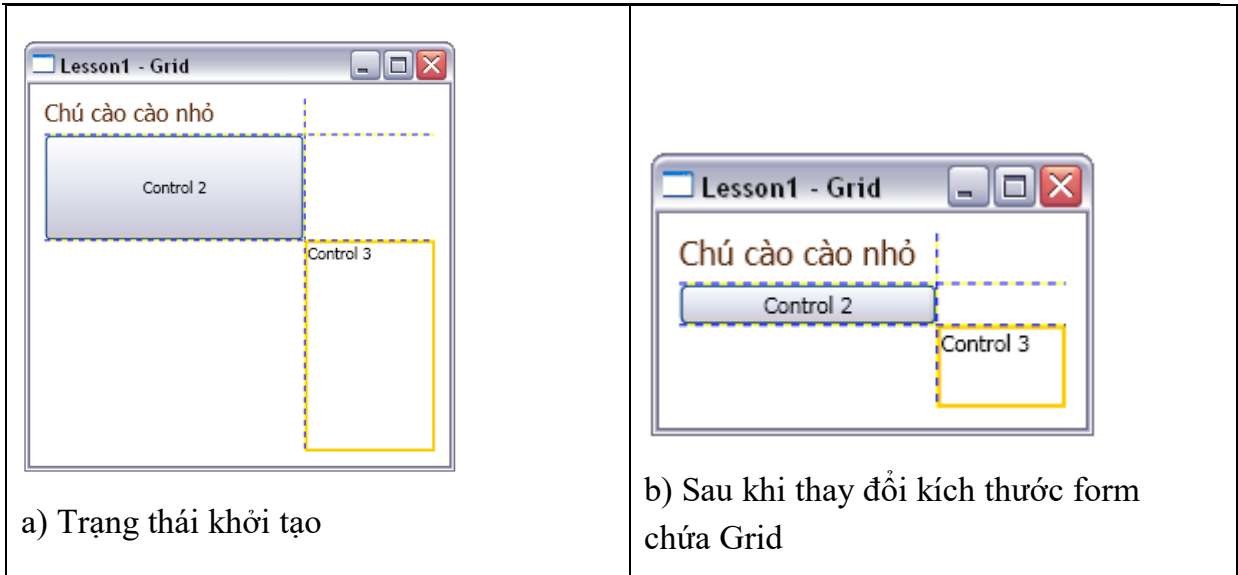
### Grid

Panel dạng **Grid** là dạng panel hết sức linh hoạt, và có thể sử dụng để đạt được gần như tất cả khả năng mà các dạng panel khác có thể làm được, mặc dù mức độ khó dễ không giống nhau. Grid cho phép ta phân định các dòng và cột theo dạng một lưới kẻ ô, và sau đó sẽ sắp đặt các phần tử UI vào các ô tùy ý. Grid sẽ tự động chia đều các dòng và cột

(dựa trên kích thước của phần nội dung). Tuy nhiên, ta có thể sử dụng dấu sao (\*) để phân định kích thước theo tỉ lệ hoặc phân định giá trị tuyệt đối về chiều cao hoặc chiều rộng cho hàng và cột. Ta có thể nhận biết sự khác biệt của 2 dạng phân định kích thước nêu trên bằng cách thay đổi kích thước của form chứa panel Grid. Thêm vào đó, thuộc tính ShowGridLines được đặt bằng True cho phép hiển thị các đường kẻ ô. Sau đây là một ví dụ minh họa về việc sử dụng Grid với hai dạng phân định:

```
<!--Sử dụng panel Grid có các cạnh cách lề 10 pixel và có hiển thị các đường kẻ ô-->
<Grid Margin="10" ShowGridLines="True">
  <!--Định nghĩa thuộc tính cột - Có tổng cộng 2 cột-->
  <Grid.ColumnDefinitions>
    <!--Khai báo cột 0 - có chiều rộng tỉ lệ gấp đôi cột kế tiếp-->
    <ColumnDefinition Width="2*" />
    <!--Khai báo cột 1 - Thuộc tính ngầm định-->
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <!--Định nghĩa thuộc tính hàng - Có tổng cộng 3 hàng -->
  <Grid.RowDefinitions>
    <!--Khai báo hàng 0 có chiều cao bằng 25 pixel-->
    <RowDefinition Height="25" />
    <!--Khai báo hàng 1 - Thuộc tính ngầm định -->
    <RowDefinition />
    <!--Khai báo hàng 2 - Đặt chiều cao gấp đôi hàng trước (hàng 1)-->
    <RowDefinition Height="2*" />
  </Grid.RowDefinitions>
  <!--Đặt TextBlock 1 vào cột 0 hàng 0-->
  <TextBlock FontSize="16"
    Foreground="#58290A"
    Grid.Column="0" Grid.Row="0">
    Chú cào cào nhỏ
  </TextBlock>
  <!--Đặt Button vào vị trí cột 0 hàng 1-->
  <Button Grid.Column="0" Grid.Row="1">
    Control 2
  </Button>
  <!--Đặt Border vào vị trí cột 1 hàng 2-->
  <Border BorderBrush="#feca00" BorderThickness="2"
    Grid.Column="1" Grid.Row="2">
    <TextBlock>Control 3</TextBlock>
  </Border>
</Grid>
```

Sau đây là kết quả:



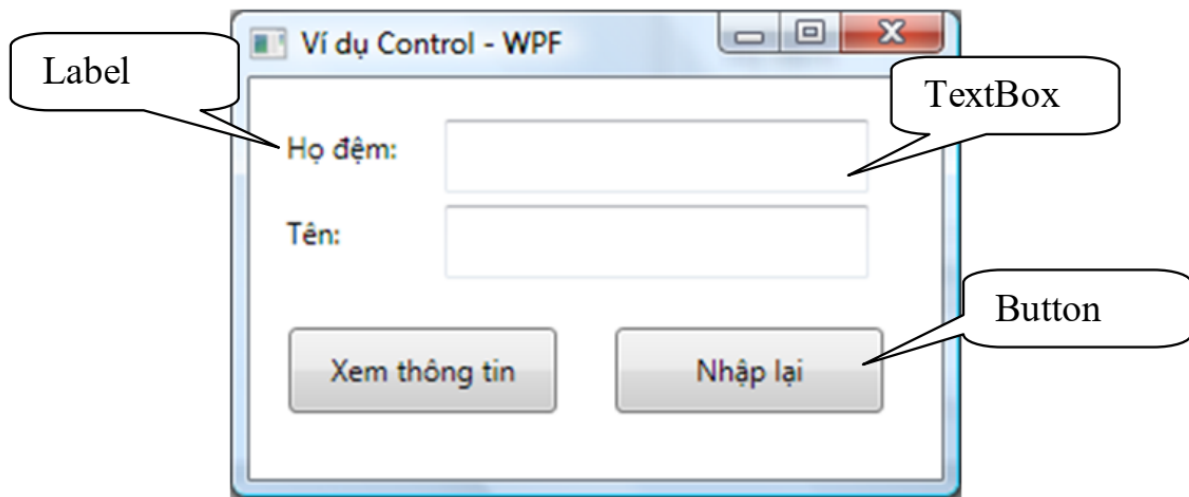
Hình 9.17 – Sử dụng Grid để sắp xếp các phần tử UI – Khi kích thước form chứa Grid thay đổi, chiều cao của hàng 1 luôn cố định (25pixel), trong khi đó, chiều cao của hàng 2 luôn gấp đôi hàng 1; chiều rộng của cột 0 luôn gấp đôi chiều rộng của cột 1.

#### 4. Các điều khiển (control) cơ bản của WPF

Trong lập trình giao diện người dùng, điều khiển (Control) là các nhân tố quan trọng cấu thành nên giao diện người dùng, cho phép họ giao tiếp với ứng dụng. Control có thể được hiểu một cách đơn giản là các phần tử trên một cửa sổ như các nhãn (Label), hộp soạn thảo (TextBox), nút bấm (Button), hộp danh sách (ListBox, ComboBox),... để hiển thị các thông tin tới người dùng và cho phép người dùng nhập thông tin cần thiết cho chương trình. Phần này giới thiệu cách tạo lập và sử dụng các Control cơ bản nhất của cửa sổ xây dựng bằng công nghệ WPF.

##### 4.1. Tổng quan về tạo lập các điều khiển với WPF

Điểm khác biệt cơ bản giữa mã lệnh tạo giao diện dựa trên WPF so với phương pháp cũ là ứng dụng WPF sử dụng các đặc tả XAML (ngoài việc sử dụng mã lệnh C# hay VB.Net) để định nghĩa giao diện, trong khi phương pháp cũ phải sử dụng trực tiếp mã lệnh của C# hay VB.Net để định nghĩa giao diện. Ví dụ, để xây dựng giao diện cửa sổ đơn giản như Hình 9.18 dưới đây.



Hình 9.18: Một ví dụ về cửa sổ với các control đơn giản

Đoạn mã trình bằng XAML:

```
<Grid>
<Label Height="30" HorizontalAlignment="Left" Margin="10,15,0,0" Name="label1"
VerticalAlignment="Top" Width="60">Họ đệm:</Label>
<Label Height="30" HorizontalAlignment="Left" Margin="10,50,0,0" Name="label2"
VerticalAlignment="Top" Width="60">Tên:</Label>
<TextBox Height="30" Margin="80,17,30,0" Name="textBox1" VerticalAlignment="Top" />
<TextBox Height="30" Margin="80,52,30,0" Name="textBox2" VerticalAlignment="Top" />
<Button Height="35" HorizontalAlignment="Left" Margin="16,0,0,27" Name="button1"
VerticalAlignment="Bottom" Width="110">Xem thông tin</Button>
<Button Height="35" HorizontalAlignment="Right" Margin="0,0,24,27" Name="button2"
VerticalAlignment="Bottom" Width="110">Nhập lại</Button>
</Grid>
```

Đoạn mã trình bằng C#:

```
// Tạo nhãn Họ đệm
this.label1 = new System.Windows.Forms.Label();
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(17, 16);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(48, 13);

this.label1.TabIndex = 0;
this.label1.Text = "Họ đệm:";
//Tạo nhãn Tên
this.label2 = new System.Windows.Forms.Label();
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(17, 50);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(29, 13);
this.label2.TabIndex = 0;
this.label2.Text = "Tên:";
//Tạo TextBox nhập Họ đệm
this.textBox1 = new System.Windows.Forms.TextBox();
this.textBox1.Location = new System.Drawing.Point(100, 16);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(160, 20);
this.textBox1.TabIndex = 1;
//Tạo TextBox nhập Tên
this.textBox2 = new System.Windows.Forms.TextBox();
```

```
this.textBox2.Location = new System.Drawing.Point(100, 50);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(160, 20);
this.textBox2.TabIndex = 1;
// Tạo nút bấm Xem thông tin
this.button1 = new System.Windows.Forms.Button();
this.button1.Location = new System.Drawing.Point(20, 114);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(99, 38);
this.button1.TabIndex = 2;
this.button1.Text = "Xem thông tin";
this.button1.UseVisualStyleBackColor = true;
//Tạo nút bấm nhập lại
this.button2 = new System.Windows.Forms.Button();
this.button2.Location = new System.Drawing.Point(161, 114);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(99, 38);
this.button2.TabIndex = 2;
this.button2.Text = "Nhập lại";
this.button2.UseVisualStyleBackColor = true;
```

Như vậy, điều chúng ta cần là tìm hiểu các thẻ XAML để mô tả các Control cần thiết. Tuy nhiên bạn không cần phải lo lắng nếu như chưa quen với các mã lệnh XAML (dựa trên XML) này vì bộ công cụ Visual Studio.Net 2008 đã hỗ trợ thiết kế giao diện trực quan và tự động sinh mã XAML tương ứng.

## 4.2. Các điều khiển cơ bản trong WPF

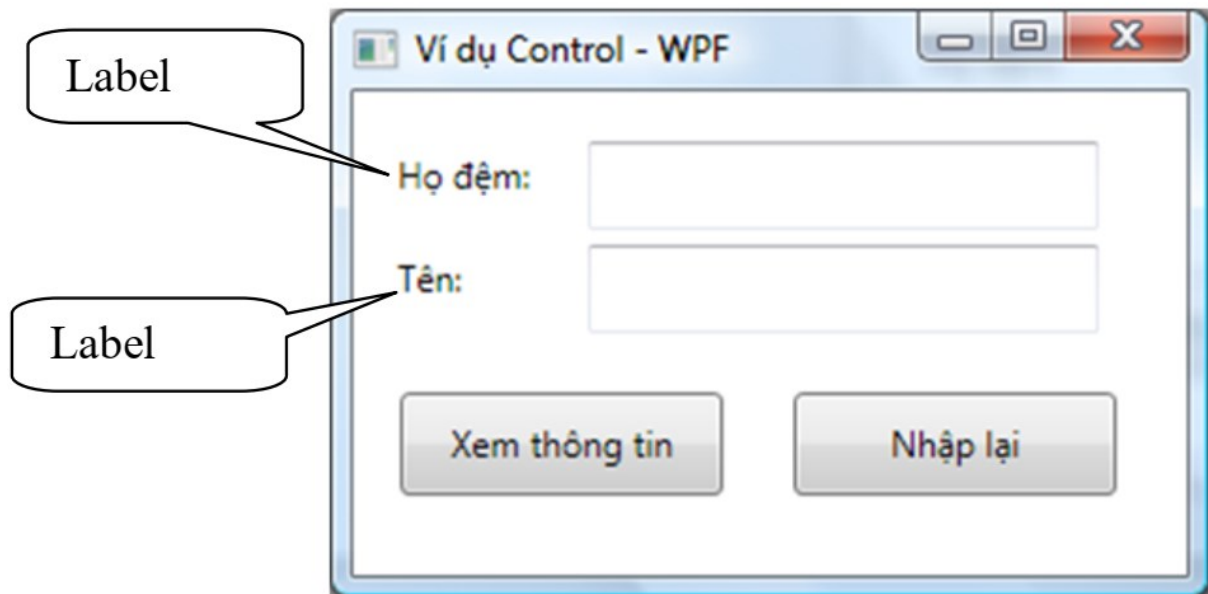
Chúng ta sẽ tìm hiểu chi tiết một số điều khiển cơ bản của cửa sổ:

- Label: Nhãn.
- TextBox: Hộp soạn thảo.
- Button: Nút bấm.
- CheckBox: Hộp chọn.
- RadioButton: Hộp chọn radio (chỉ được phép chọn 1 mục trong mỗi nhóm).
- ListBox: Hộp danh sách
- ComboBox: Hộp danh sách thả xuống..

### 4.2.1. LABEL – Nhãn

Nhãn (Label) là các điều kiện để hiển thị các văn bản tĩnh, thường được sử dụng để làm nhãn cho các control khác như Textbox, ListBox, ComboBox,... .





Hình 9.18: Minh họa về label

Các Label được mô tả bằng đoạn mã XAML sau:

```
<Grid>
<Label Height="30" HorizontalAlignment="Left" Margin="10,15,0,0" Name="label1"
VerticalAlignment="Top" Width="60">Họ đệm:</Label>
</Grid>
```

Nhãn được bắt đầu `<Label>` và kết thúc là `</Label>`, nội dung của nhãn là đoạn văn bản đặt giữa cặp thẻ này. Trong ví dụ này “Họ đệm:” là nội dung của nhãn.

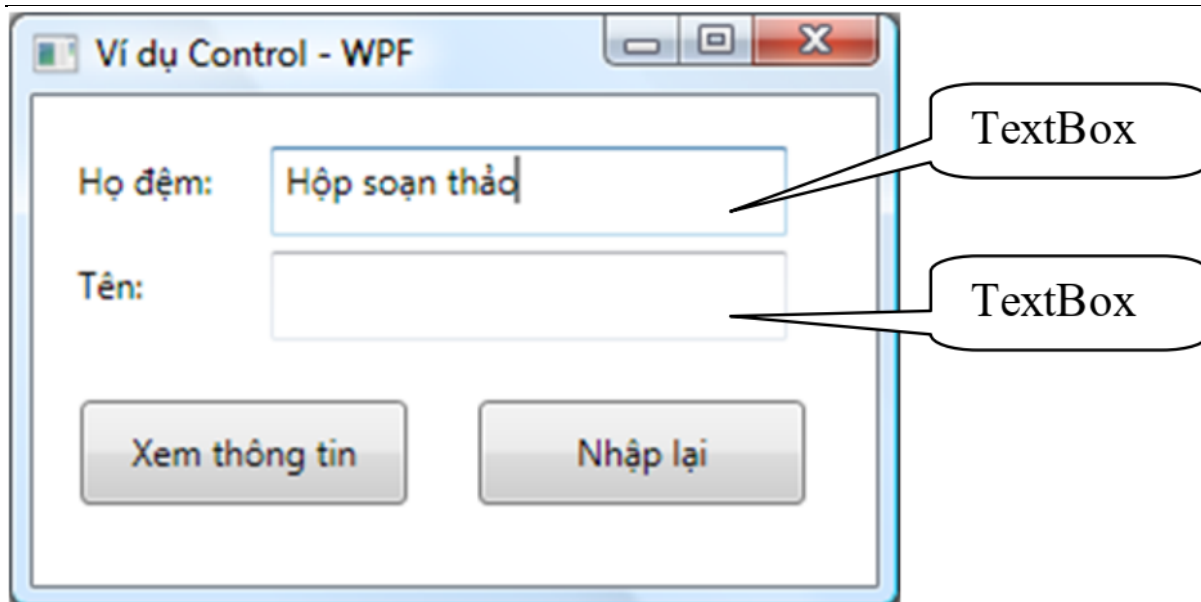
Bên trong thẻ `<Label>` có rất nhiều đặc tính để mô tả về thẻ, trong đó:

- `Height="30"` : Độ cao của khung nhãn là 30px
- `HorizontalAlignment="Left"` : Nhãn được căn trái trong cửa sổ
- `Margin="10,15,0,0"` : có 4 giá trị là Left,Top,Right,Bottom
- `Name="label1"` : Tên của nhãn là label1
- `VerticalAlignment="Top"` : Nhãn được căn theo đỉnh của cửa sổ.
- `Width="60"` : Chiều rộng của nhãn là 60px

Trên đây là một số đặc tính cơ bản của nhãn, ngoài ra còn có nhiều đặc tính khác áp dụng cho nhãn như màu nền, màu chữ,....

#### 4.2.2. TextBox – Hộp soạn thảo

Hộp soạn thảo (TextBox) là control cho phép người dùng nhập dữ liệu dạng văn bản.



Hình 9.19: Minh họa về textbox

Dưới đây là đoạn mã XAML của hộp soạn thảo

```
<Grid>
<TextBox Height="30" Margin="80,17,30,0" Name="textBox1" VerticalAlignment="Top" />
Hộp soạn thảo
</TextBox>
<TextBox Height="30" Margin="80,52,30,0" Name="textBox2" VerticalAlignment="Top" />
</Grid>
```

Hộp soạn thảo được tạo nên bởi thẻ `<TextBox/>`. Nếu muốn thiết lập sẵn nội dung mặc định cho hộp soạn thảo, ta đặt nội dung này vào giữa cặp thẻ `<TextBox/>` *Nội dung* `</TextBox>`. Nếu không muốn đặt giá trị mặc định thì không cần thẻ đóng `</TextBox>`.

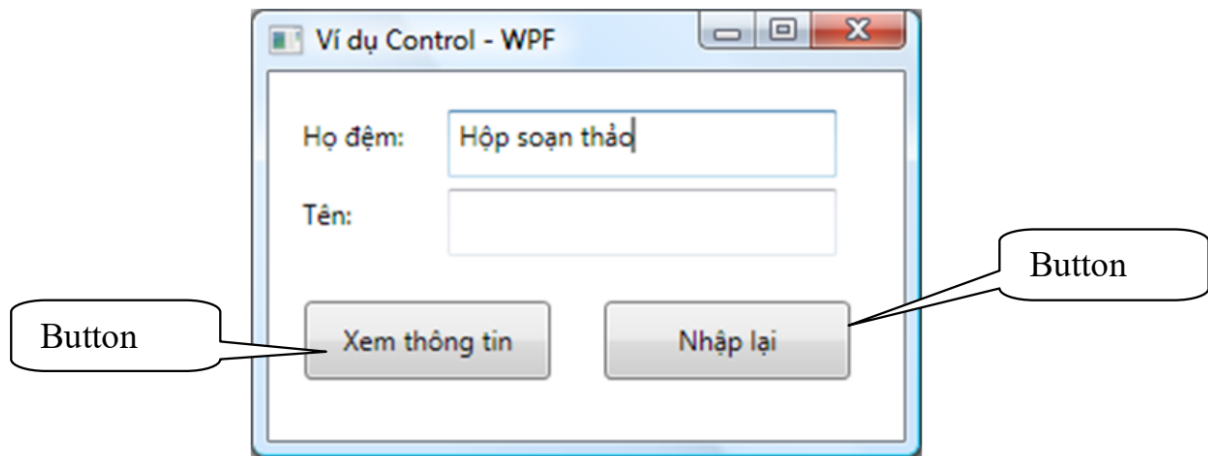
Thẻ `<TextBox/>` cũng có nhiều đặc tính, trong đó:

- `Margin="80,17,30,0"`: Cách lề trái 80, đỉnh cửa sổ 17, cạnh phải 30
- `Name="textBox1"`: Tên của hộp soạn thảo là textBox1
- `VerticalAlignment="Top"`: Căn theo đỉnh cửa sổ

Đặc điểm của hộp soạn thảo với các đặc tính trên là khi người dùng co dãn, thay đổi kích thước cửa sổ, chiều rộng của hộp soạn thảo tự động co dãn theo.

#### 4.2.3. Button – Nút bấm

Nút bấm (Button) là loại điều khiển cho phép người dùng nhấn chuột để chọn lệnh, khi nhấn vào nút bấm, nó sẽ sinh ra sự kiện *Click* và sẽ chạy các lệnh gắn với sự kiện này.



Hình 9.20: Minh họa về nút bấm

Dưới đây là đoạn mã sinh ra các nút bấm trên

```
<Grid>
<Button Height="35" HorizontalAlignment="Left" Margin="16,0,0,27" Name="button1"
VerticalAlignment="Bottom" Width="110" Click="button1_Click">Xem thông tin</Button>
<Button Height="35" HorizontalAlignment="Right" Margin="0,0,24,27" Name="button2"
VerticalAlignment="Bottom" Width="110">Nhập lại</Button>
</Grid>
```

Nút bấm được bắt đầu bằng thẻ `<Button>` và kết thúc bằng thẻ `</Button>`. Nhãn của nút bấm được đặt trong cặp thẻ `<Button> Nhãn nút bấm </Button>`.

Nút bấm có nhiều đặc tính, trong đó:

- `Height="35"`: Chiều cao nút bấm là 35
- `Width="110"`: Chiều rộng là 110
- `HorizontalAlignment="Left"`: Căn theo lề trái
- `VerticalAlignment="Bottom"`: Căn theo đáy cửa sổ
- `Margin="16,0,0,27"`: Cách lề trái 16, cách đáy 27
- `Name="button1"`: Tên nút bấm là button1
- `Click="button1_Click"`: Khi nhấn chuột vào nút sẽ kích hoạt phương thức `button1_Click()`

Ví dụ về đoạn mã lệnh gắn với cửa sổ chứa các điều khiển trên:

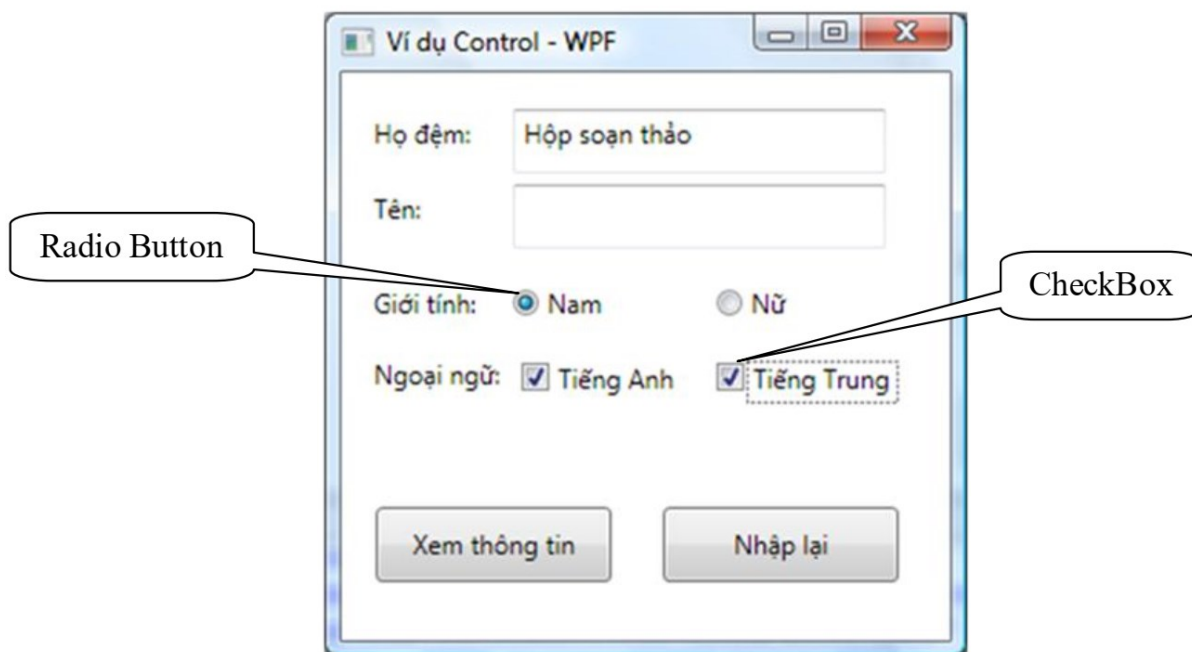
```
namespace WpfControlSample1
{
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, RoutedEventArgs e)
        {
            String strHoTen;
            strHoTen = textBox1.Text + " " + textBox2.Text;
            MessageBox.Show("Xin chào: " + strHoTen);
        }
    }
}
```

#### 4.2.4. Radio Button và CheckBox

Radio Button và CheckBox đều là điều khiển dạng hộp chọn. Tuy nhiên, điểm khác biệt cơ bản giữa hai loại điều khiển này là:

**Radio Button:** là hộp chọn theo nhóm, nghĩa là các hộp trong cùng một nhóm sẽ loại trừ nhau, tại một thời điểm người dùng chỉ được chọn một trong các mục. Ví dụ như hộp chọn giới tính, ta phải sử dụng radio vì tại một thời điểm chỉ cho phép chọn Nam hoặc Nữ.

**CheckBox:** là hộp chọn mà người dùng có thể chọn một hoặc nhiều mục cùng một lúc. Ví dụ như mục chọn Ngoại ngữ, cho phép người dùng chọn đồng thời nhiều mục.



Hình 9.21: Minh họa về hộp chọn Radio và CheckBox

Dưới đây là đoạn mã tạo Radio Button và CheckBox.

```
<Grid>
<Label Height="30" HorizontalAlignment="Left" Margin="10,94,0,0" Name="label3"
VerticalAlignment="Top" Width="60">Giới tính:</Label>
<RadioButton Height="22" Margin="80,99,0,0" Name="radioButton1"
VerticalAlignment="Top" HorizontalAlignment="Left" Width="79" GroupName="GioiTinh"
IsChecked="True">Nam</RadioButton>
<RadioButton Height="22" HorizontalAlignment="Right" Margin="0,99,30,0"
Name="radioButton2" VerticalAlignment="Top" Width="79"
GroupName="GioiTinh">Nữ</RadioButton>
<Label HorizontalAlignment="Left" Margin="10,127,0,105" Name="label4"
Width="69">Ngoại ngữ:</Label>
<CheckBox Margin="84,0,119,110" Name="checkBox1" Height="20"
VerticalAlignment="Bottom" IsChecked="True">Tiếng Anh</CheckBox>
<CheckBox HorizontalAlignment="Right" Margin="0,0,24,110" Name="checkBox2"
Width="85" Height="20" VerticalAlignment="Bottom">Tiếng Trung</CheckBox>
</Grid>
```

Radio Button được tạo bởi thẻ `<RadioButton>` và kết thúc bởi `</RadioButton>`, giữa cặp thẻ này là nhãn của Radio Button `<RadioButton> Nhãn </RadioButton>`.

CheckBox được tạo bởi thẻ `<CheckBox>` và kết thúc bởi `</CheckBox>`, giữa cặp thẻ này là nhãn của CheckBox `<CheckBox> Nhãn </CheckBox>`. Cả hai thẻ này đều có đặc tính `IsChecked="True"` hoặc `IsChecked="False"`. Mục nào có thuộc tính này sẽ được tự động chọn khi cửa sổ bắt đầu hiển thị. Đối với Radio Button, vì là hộp chọn loại trừ, nếu trong một cửa sổ có nhiều nhóm Radio Button khác nhau thì các Radio Button của mỗi nhóm được phân biệt bởi đặc tính `GroupName="TenNhóm"`. Ví dụ, trên cùng một cửa sổ có hai Radio Button chọn Giới tính (Nam; Nữ) và ba Radio Button khác chọn nghề nghiệp (Kinh doanh; Kỹ Thuật; Marketing) thì các Radio Button Nam, Nữ phải có cùng GroupName với nhau, ba Radio Button Kinh doanh, Kỹ Thuật, Marketing phải có cùng GroupName và khác với GroupName của nhóm giới tính.

Đoạn mã lệnh minh họa kiểm tra mục chọn Radio

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    String strMessage, strHoTen, strTitle, strNgoaiNgu="";
    strHoTen = textBox1.Text + " " + textBox2.Text;
    if (radioButton1.IsChecked==true)
        strTitle = "Mr.";
    else
        strTitle = "Miss/Mrs.";
    strMessage = "Xin chào: " + strTitle + " " + strHoTen;
    MessageBox.Show(strMessage);
}
```

Đoạn mã lệnh minh họa kiểm tra mục chọn CheckBox

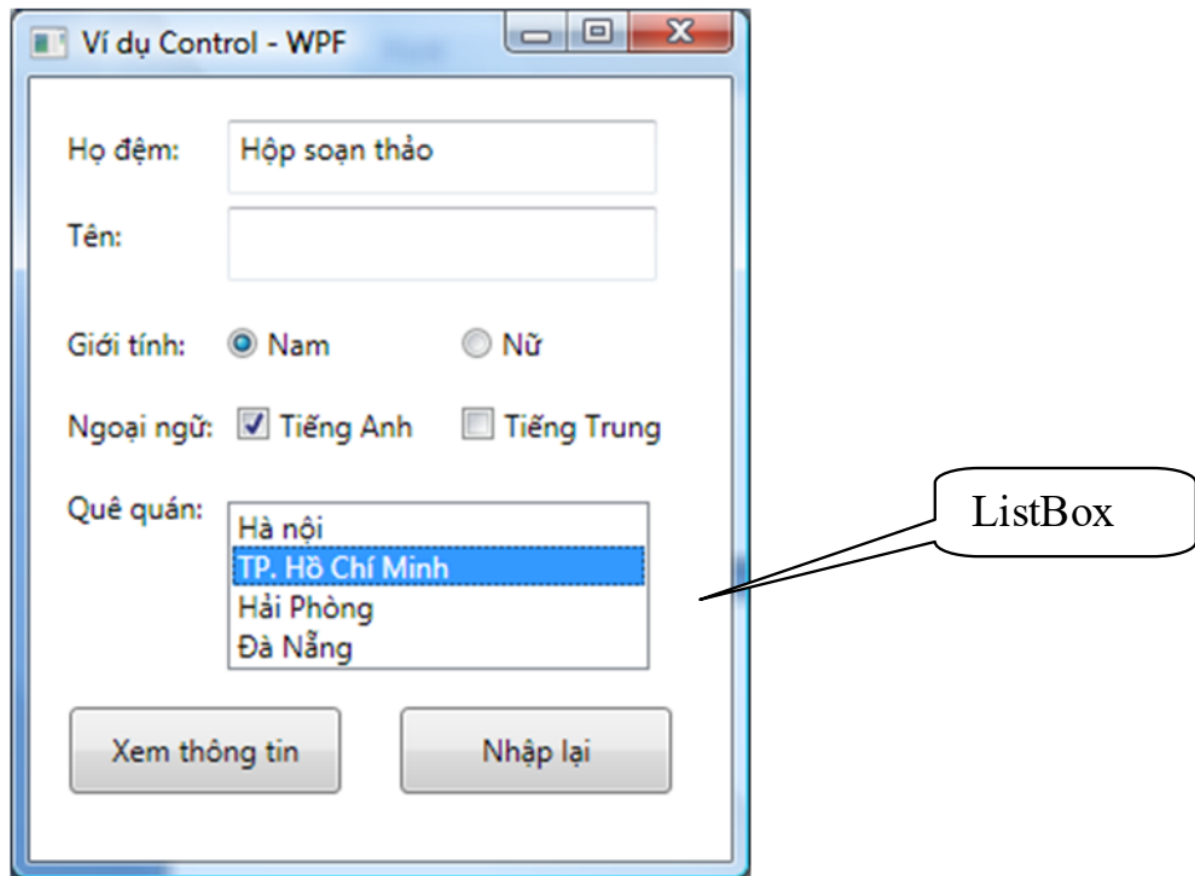
```
private void button1_Click(object sender, RoutedEventArgs e)
{
    String strMessage, strNgoaiNgu="";
    if (checkBox1.IsChecked == true)
    {
        strNgoaiNgu = "Tiếng Anh";
    }
    if (checkBox2.IsChecked == true)
    {
        strNgoaiNgu = (strNgoaiNgu.Length==0) ? "Tiếng Trung": (strNgoaiNgu+ " và Tiếng Trung");
    }
    strMessage = "Ngoại ngữ: " + strNgoaiNgu;
    MessageBox.Show(strMessage);
}
```

#### 4.2.5. ListBox - Hộp danh sách

Hộp danh sách (ListBox) và là điều khiển hiển thị một danh sách các mục theo từng dòng và cho phép người dùng chọn một hay nhiều phần tử của danh sách.

Ví dụ về hộp danh sách chọn Quê quán:





Hình 9.22: Minh họa về hộp chọn ListBox

Dưới đây là đoạn mã tạo ListBox

```
<Grid>
<Label Height="30" HorizontalAlignment="Left" Margin="10,0,0,126" Name="label5"
VerticalAlignment="Bottom" Width="69">Quê quán:</Label>
<ListBox Height="68" Margin="80,0,33,77" Name="listBox1" VerticalAlignment="Bottom"
SelectedIndex="0">
<ListBoxItem>Hà nội</ListBoxItem>
<ListBoxItem>TP. Hồ Chí Minh</ListBoxItem>
<ListBoxItem>Hải Phòng</ListBoxItem>
<ListBoxItem>Đà Nẵng</ListBoxItem>
</ListBox>
</Grid>
```

ListBox được tạo bởi thẻ `<ListBox>` và kết thúc bằng thẻ đóng `</ListBox>`.

Mỗi phần tử của danh sách nằm trong cặp thẻ `<ListBoxItem>` *Nhãn* `</ListBoxItem>` lồng bên trong cặp thẻ trên.

Đặc tính `SelectedIndex="k"` để yêu cầu tự động chọn phần tử thứ  $n$  trong danh sách khi mở cửa sổ. Phần tử đầu tiên của danh sách có giá trị là 0, phần tử cuối cùng là  $n-1$ . Nếu muốn khi mở cửa sổ không chọn phần tử nào thì đặt giá trị  $k$  bằng -1.

Đoạn mã lệnh lấy mục chọn từ ListBox

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    String strMessage;

    if (listBox1.SelectedIndex >= 0) //Nếu đã có một mục trong
    {
        danh sách được chọn
    }
}
```

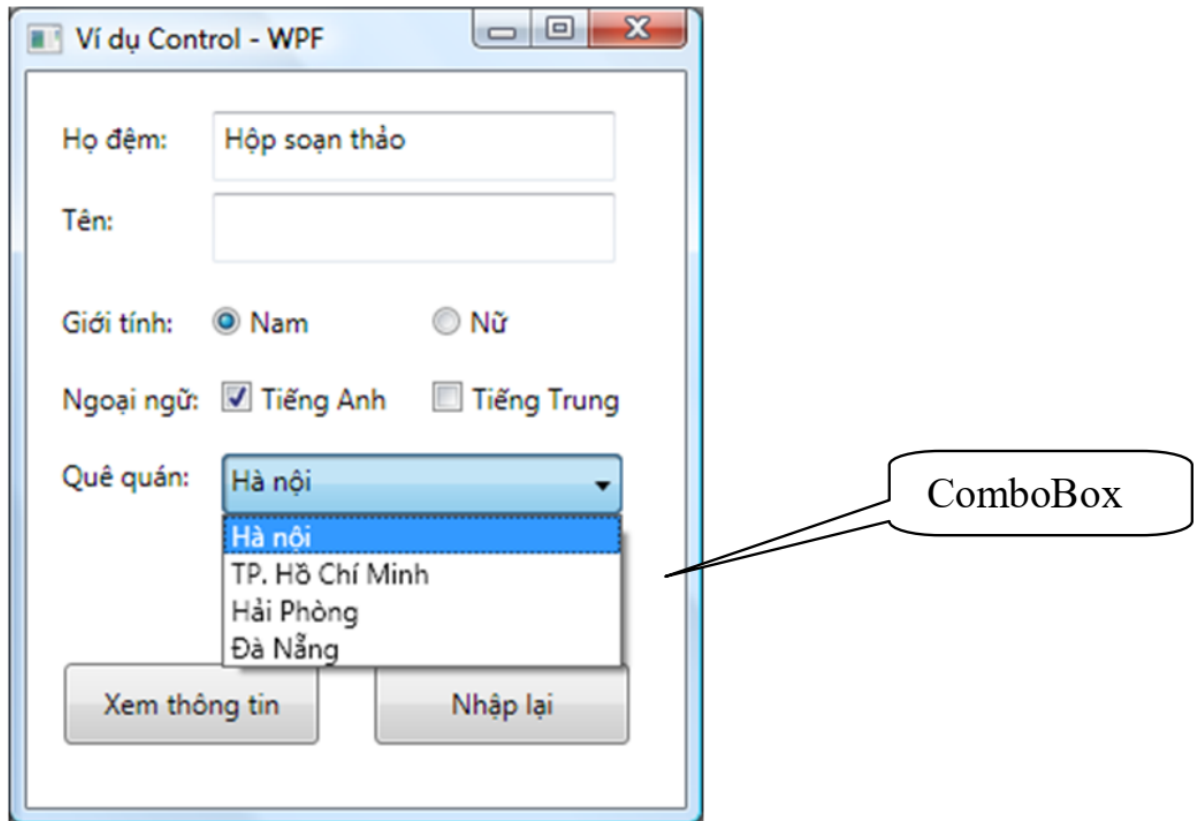
```

strMessage = "Quê Quán: " + listBox1.Text;
}
MessageBox.Show(strMessage);
}

```

#### 4.2.6. ComboBox - hộp danh sách thả xuống

Hộp danh sách thả xuống (ComboBox) là các điều khiển hiển thị một danh sách theo từng dòng cho người dùng chọn. Tuy nhiên, khác với ListBox, ComboBox gọn gàng hơn bởi vì nó chỉ hiển thị 1 dòng và khi nhấn vào biểu tượng tam giác bên cạnh thì danh sách mới được mở ra. Combox chỉ cho phép chọn 1 dòng tại 1 thời điểm. Ví dụ về hộp danh sách thả xuống chọn Quê quán:



Hình 9.23: Minh họa về hộp chọn ComboBox

Dưới đây là đoạn mã tạo ComboBox

```

<Grid>
<Label Height="30" HorizontalAlignment="Left" Margin="10,0,0,126" Name="label5"
VerticalAlignment="Bottom" Width="69">Quê quán:</Label>
<ComboBox Height="26" Margin="84,0,27,126" Name="comboBox1"
VerticalAlignment="Bottom" SelectedIndex="0">
<ComboBoxItem>Hà nội</ComboBoxItem>
<ComboBoxItem>TP. Hồ Chí Minh</ComboBoxItem>
<ComboBoxItem>Hải Phòng</ComboBoxItem>
<ComboBoxItem>Đà Nẵng</ComboBoxItem>
</ComboBox>
</Grid>

```

ComboBox được tạo bởi thẻ `<ComboBox>` và kết thúc bằng thẻ đóng `</ComboBox>`. Mỗi phần tử của danh sách nằm trong cặp thẻ `<ComboBoxItem>` `</ComboBoxItem>` Nằm bên trong cặp thẻ trên.

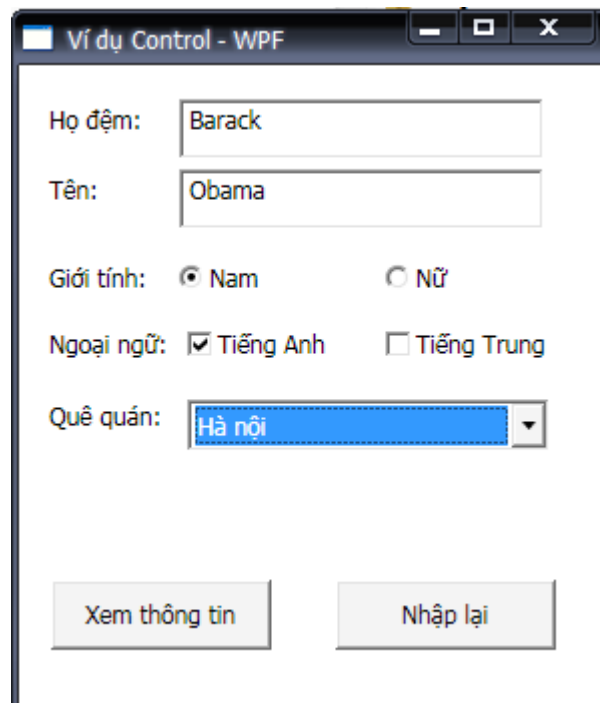
Đặc tính *SelectedIndex*="k" để yêu cầu tự động chọn phần tử thứ n trong danh sách khi mở cửa sổ. Phần tử đầu tiên của danh sách có giá trị là 0, phần tử cuối cùng là n-1. Nếu muốn khi mở cửa sổ không chọn phần tử nào thì đặt giá trị k bằng -1.

Đoạn mã lệnh lấy mục chọn từ ComboBox

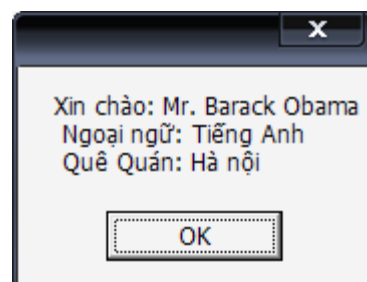
```
private void button1_Click(object sender, RoutedEventArgs e)
{
    String strMessage;
    if (comboBox1.SelectedIndex >= 0)//Nếu đã có một mục trong danh sách được chọn
    {
        strMessage = "Quê Quán: " + comboBox1.Text;
    }
    MessageBox.Show(strMessage);
}
```

### 4.3. Ví dụ xây dựng các control trong WPF

Yêu cầu: Viết chương trình hiển thị cửa sổ như sau



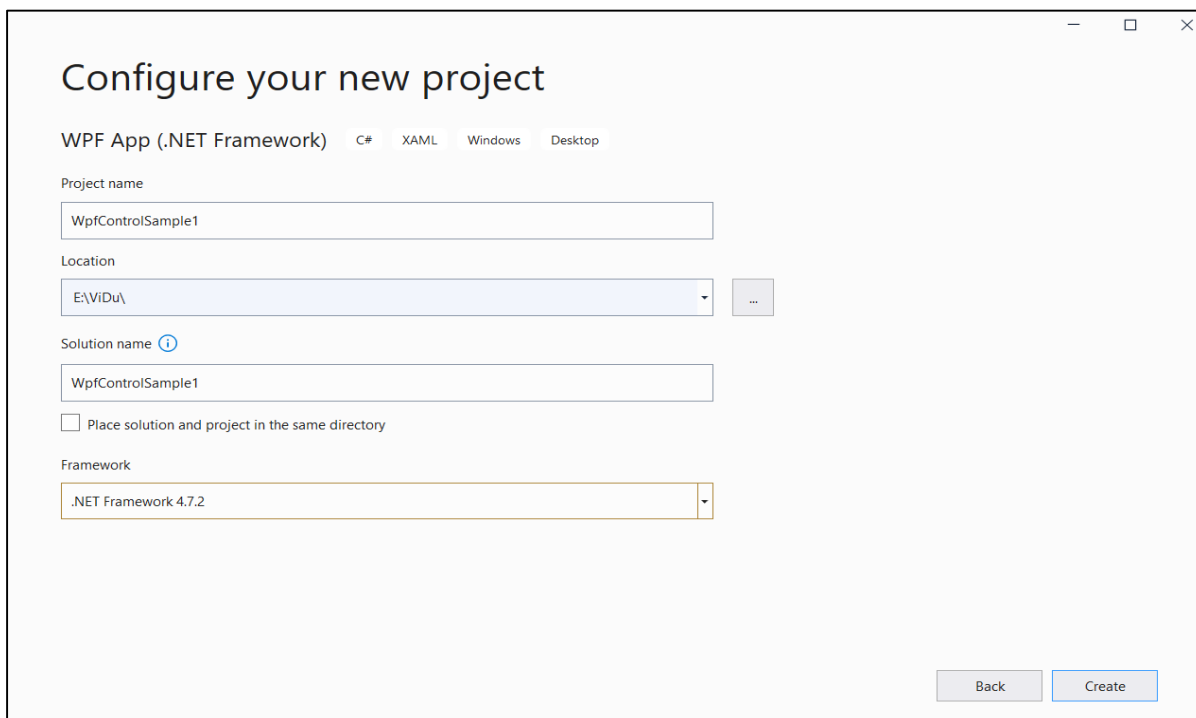
Khi nhấn Xem thông tin, nếu chọn Nam thì hiện lời chào “Xin chào Mr. Barack Obama”, nếu chọn Nữ thì hiện lời chào “Xin chào Miss/Mrs. Barack Obama”, như sau:



Nếu chọn Nhập lại, đưa trạng thái các control trên cửa sổ về trạng thái ban đầu.

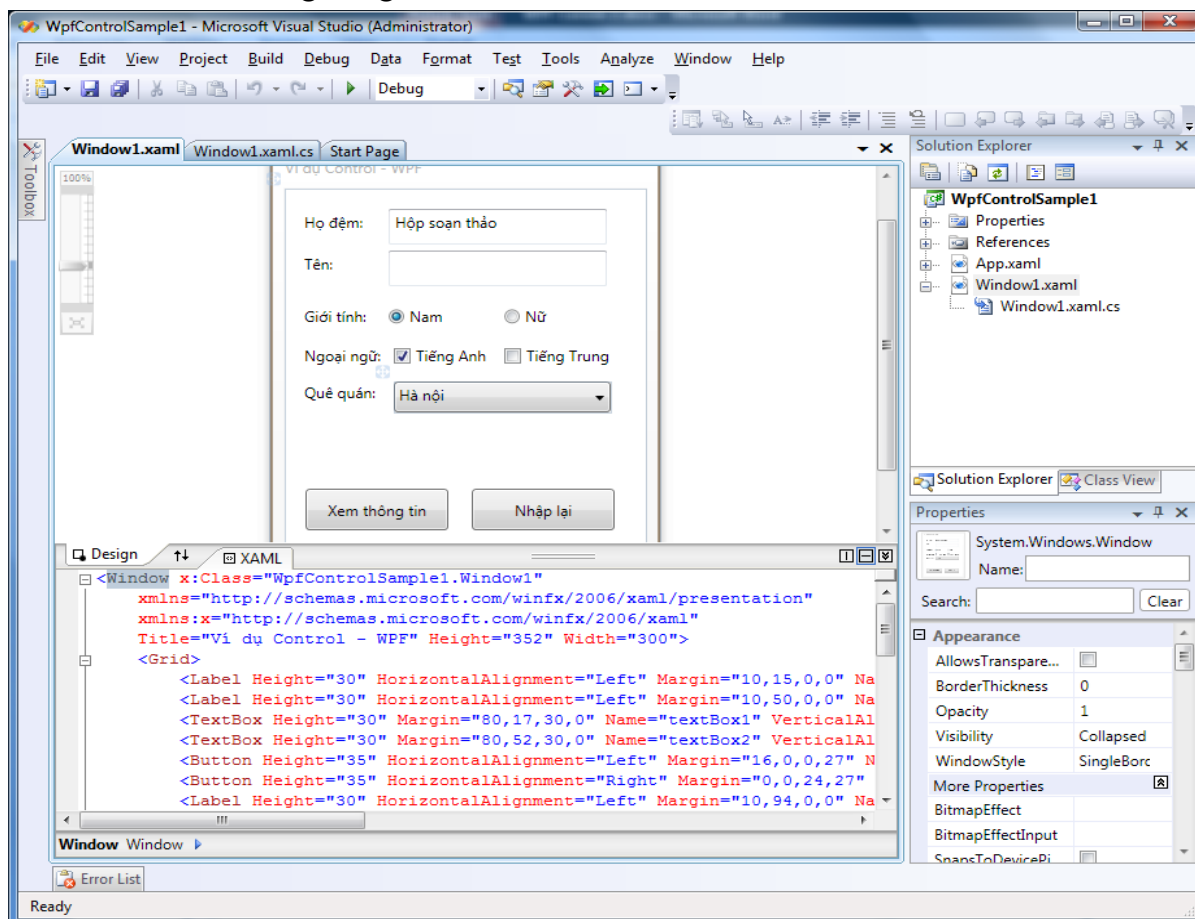
Các bước thực hiện như sau:

### 4.3.1 Tạo ứng dụng WPF



Ở đây, ta chú ý chọn .Net Framework 5.0

Giao diện thiết kế ứng dụng:



### 4.3.2 Mã XAML của giao diện

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

```
<Window x:Class="WpfControlSample1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Ví dụ Control - WPF" Height="352" Width="300">
<Grid>
<Label Height="30" HorizontalAlignment="Left" Margin="10,15,0,0" Name="label1"
VerticalAlignment="Top" Width="60" Focusable="False">Họ đệm:</Label>
<Label Height="30" HorizontalAlignment="Left" Margin="10,50,0,0" Name="label2"
VerticalAlignment="Top" Width="60">Tên:</Label>
<TextBox Height="30" Margin="80,17,30,0" Name="textBox1"
VerticalAlignment="Top">Hộp soạn thảo</TextBox>
<TextBox Height="30" Margin="80,52,30,0" Name="textBox2" VerticalAlignment="Top" />
<Button Height="35" HorizontalAlignment="Left" Margin="16,0,0,27" Name="button1"
VerticalAlignment="Bottom" Width="110" Click="button1_Click">Xem thông tin</Button>
<Button Height="35" HorizontalAlignment="Right" Margin="0,0,24,27" Name="button2"
VerticalAlignment="Bottom" Width="110" Click="button2_Click">Nhập lại</Button>
<Label Height="30" HorizontalAlignment="Left" Margin="10,94,0,0" Name="label3"
VerticalAlignment="Top" Width="60">Giới tính:</Label>
<RadioButton Height="22" Margin="80,99,0,0" Name="radioButton1"
VerticalAlignment="Top" HorizontalAlignment="Left" Width="79" GroupName="GioiTinh"
IsChecked="True">Nam</RadioButton>
<RadioButton Height="22" HorizontalAlignment="Right" Margin="0,99,30,0"
Name="radioButton2" VerticalAlignment="Top" Width="79"
GroupName="GioiTinh">Nữ</RadioButton>
<Label HorizontalAlignment="Left" Margin="10,127,0,154" Name="label4"
Width="69">Ngoại ngữ:</Label>
<CheckBox Margin="84,132,119,0" Name="checkBox1" Height="20"
VerticalAlignment="Top" IsChecked="True">Tiếng Anh</CheckBox>
<CheckBox HorizontalAlignment="Right" Margin="0,132,24,0" Name="checkBox2"
Width="85" Height="20" VerticalAlignment="Top">Tiếng Trung</CheckBox>
<Label Height="30" HorizontalAlignment="Left" Margin="10,0,0,126" Name="label5"
VerticalAlignment="Bottom" Width="69">Quê quán:</Label>
<ComboBox Height="26" Margin="84,0,27,126" Name="comboBox1"
VerticalAlignment="Bottom" SelectedIndex="0">
<ComboBoxItem>Hà nội</ComboBoxItem>
<ComboBoxItem>TP. Hồ Chí Minh</ComboBoxItem>
<ComboBoxItem>Hải Phòng</ComboBoxItem>
<ComboBoxItem>Đà Nẵng</ComboBoxItem>
</ComboBox>
</Grid>
</Window>
```

Gán tên phương thức *button1\_Click* xử lý sự kiện Click cho nút bấm *button1* này, khi nhấn chuột vào nút “Xem thông tin” thì phương thức *button1\_Click* sẽ được thực thi.

```
<Button Height="35" HorizontalAlignment="Left" Margin="16,0,0,27" Name="button1"
VerticalAlignment="Bottom" Width="110" Click="button1_Click">Xem thông tin</Button>
```

Gán tên phương thức *button2\_Click* xử lý sự kiện Click cho nút bấm *button2* này, khi nhấn chuột vào nút “Nhập lại” thì phương thức *button2\_Click* sẽ được thực thi.

```
<Button Height="35" HorizontalAlignment="Right" Margin="0,0,24,27" Name="button2"
VerticalAlignment="Bottom" Width="110" Click="button2_Click">Nhập lại</Button>
```

### 4.3.3. Mã lệnh C# xử lý các sự kiện nhấn nút



```
using System;
using System.Collections.Generic;
using System.Linq;
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.Navigation;
using System.Windows.Shapes;
namespace WpfControlSample1
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, RoutedEventArgs e)
        {
            String strMessage, strHoTen, strTitle, strNgoaiNgu="";
            strHoTen = textBox1.Text + " " + textBox2.Text;
            if (radioButton1.IsChecked==true)
                strTitle = "Mr.";
            else
                strTitle = "Miss/Mrs.";
            strMessage = "Xin chào: " + strTitle + " " + strHoTen;
            if (checkBox1.IsChecked == true)
            {
                strNgoaiNgu = "Tiếng Anh";
            }
            if (checkBox2.IsChecked == true)
            {
                strNgoaiNgu = (strNgoaiNgu.Length==0) ? "Tiếng Trung": (strNgoaiNgu+"
và Tiếng Trung");
            }
            strMessage += "\n Ngoại ngữ: " + strNgoaiNgu;
            if (comboBox1.SelectedIndex >= 0)//Nếu đã có một mục trong danh sách được
            chọn
            {
                strMessage += "\n Quê Quán: " + comboBox1.Text;
            }
            MessageBox.Show(strMessage);
        }
        private void button2_Click(object sender, RoutedEventArgs e)
        {
            textBox1.Text = "";
            textBox2.Text = "";
            radioButton1.IsChecked = true;
            radioButton2.IsChecked = false;
            checkBox1.IsChecked = false;
            checkBox2.IsChecked = false;
            comboBox1.SelectedIndex = 0;
        }
    }
}
```

```
}
}
```

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

## 5. Các điều khiển nâng cao trong WPF

Không dừng lại ở việc cung cấp những điều khiển UI như ComboBox, ListBox, TextBox,..., với những chức năng cơ bản và đặc tính text đơn điệu như trong Windows Form, WPF còn cho phép người lập trình tùy biến thuộc tính của những điều khiển trên để biến chúng thành những điều khiển UI phức hợp, với nhiều đặc tính giao diện phong phú, tinh tế, kết hợp text, hình ảnh,... Để đạt được hiệu quả tương tự, với những công nghệ trước đây như MFC, cần tiêu tốn nhiều công sức lập trình. Qua các ví dụ cụ thể trong bài giảng này, chúng ta sẽ thấy WPF tạo ra chúng đơn giản như thế nào.

### 5.1. Hộp lựa chọn phong chữ (Font Chooser)

Mục tiêu của phần này là tạo lập một điều khiển dạng ComboBox, trong đó, liệt kê danh sách các phong chữ hệ thống. Tên của mỗi phong chữ lại được hiển thị dưới dạng chính phong chữ đó. Điều này cho phép người dùng xem trước định dạng phong chữ trước khi chọn chúng. Bạn có thể đã quen thuộc với dạng Combox này khi sử dụng các ứng dụng gần đây của Microsoft Office như Word, Excel, PowerPoint,... Và sau đây là mã XAML để tạo ra điều khiển này:

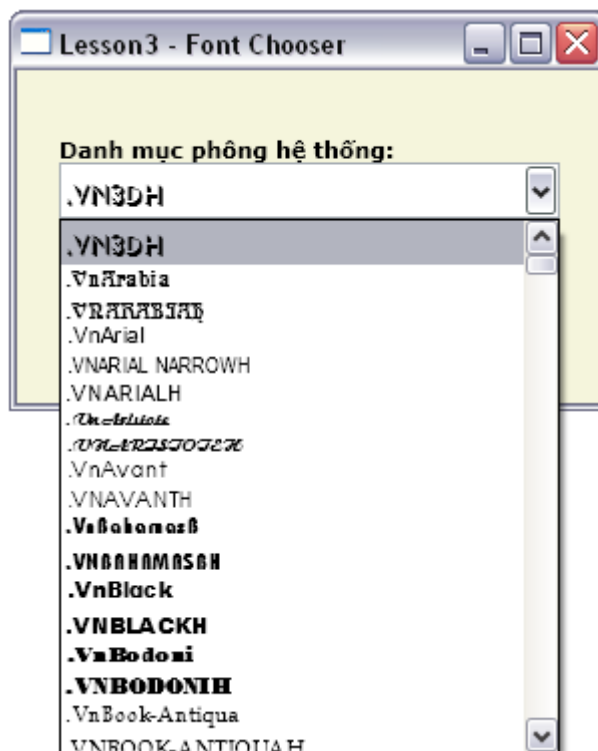
```
<!--Sử dụng stack panel theo chiều dọc làm layout chính của form-->
<StackPanel Width="250" Orientation="Vertical" Height="100" >
<!--Khai báo tạo lập tiêu đề của điều khiển-->
<TextBlock FontFamily="Verdana" FontWeight="DemiBold">
Danh mục phong hệ thống:
</TextBlock>
<!--Khai báo tạo điều khiển Combox-->
<ComboBox ItemsSource="{x:Static Fonts.SystemFontFamilies}" SelectedIndex="0">
<!--Khai báo định nghĩa thuộc tính dữ liệu gắn với mỗi mục chọn-->
<ComboBox.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding}" FontFamily="{Binding}"/>
</DataTemplate>
</ComboBox.ItemTemplate>
</ComboBox>
</StackPanel>
```

Trong phần khai báo tạo điều khiển ComboBox, ta khai báo nguồn dữ liệu được dùng cho các mục trong hộp danh sách thông qua thuộc tính **ItemsSource**. Bằng việc gán **temsSource="{x:StaticFonts.SystemFontFamilies}"** ta định nghĩa nguồn dữ liệu này là danh sách các phong chữ hiện có của hệ thống máy tính hiện thời. Thuộc tính **SelectedIndex** cho phép định ra chỉ số của chỉ mục ngầm định được chọn ban đầu trong danh sách phong, cụ thể trong trường hợp này là phong chữ đầu tiên (**SelectedIndex="0"**).

Trong phần khai báo định nghĩa thuộc tính dữ liệu của mỗi chỉ mục trong ComboBox (phần tử **<ComboBox.ItemTemplate>**), ta lồng vào một điều khiển **TextBlock**, trong đó, nội dung hiển thị là phong chữ tương ứng (**Text="{Binding}"**) và dạng phong hiển thị

nội dung cũng chính là phong chữ tương ứng với chỉ mục này (`FontFamily="{Binding}"`). Những vấn đề liên quan đến kết nối nguồn dữ liệu sẽ được đề cập chi tiết hơn trong các bài giảng tiếp sau.

Biên dịch và chạy chương trình, ta có kết quả như minh họa ở Hình 5.1. Như vậy, chỉ với không hơn 20 dòng mã XAML, chúng ta đã có thể tạo ra một điều khiển rất hữu dụng.



Hình 9.25 – Hộp lựa chọn phông chữ được xây dựng bằng WPF

## 5.2. Hộp danh mục ảnh (Image ListBox)

Trong phần này, ta xây dựng một hộp danh mục (ListBox) các đồ uống có kèm theo ảnh. Rõ ràng tính trực quan của giao diện người dùng sẽ tăng hơn nhiều so với một danh sách dạng text đơn điệu.

### 5.2.1 Thêm dữ liệu ảnh vào tài nguyên của project

Trước hết, ta thêm các ảnh đồ uống cần thiết vào tài nguyên của project theo các bước sau:

- Ở cửa sổ Solution Explorer, ta nhấp chuột phải vào tên project → Xuất hiện bảng chọn chức năng.
- Chọn mục Add...>Existing Item → Xuất hiện cửa sổ cho phép lựa chọn file.
- Trong hộp danh sách Files of type, ta chọn Image Files → Các file ảnh trong thư mục hiện thời sẽ xuất hiện.
- Tìm đến các file ảnh cần hiển thị trong danh sách và chọn OK.
- Kết quả, trong cửa sổ Solution Explorer, ta thấy xuất hiện các file ảnh tương ứng.

## 5.2.2 Xây dựng mã XAML

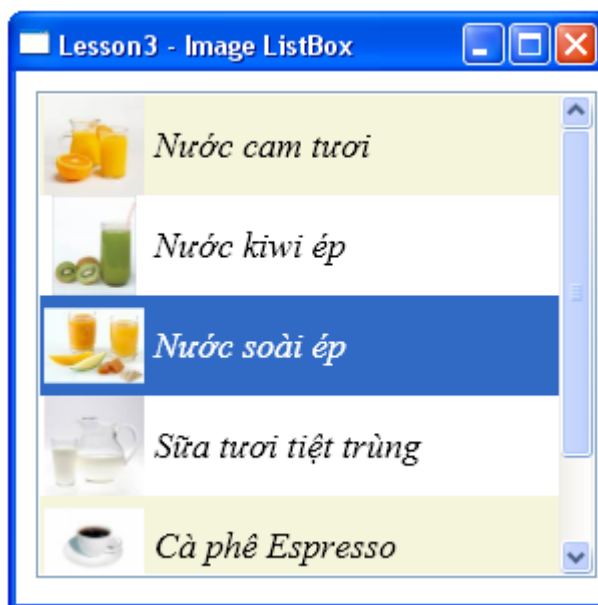
Giả thiết rằng các file ảnh đã được nạp vào project, sau đây là mã XAML tạo lập hộp danh mục đồ uống theo yêu cầu:

```
<!-- Khai báo tạo lập một hộp danh mục với các thuộc tính về căn lề, chiều rộng,...-->
<ListBox Margin="10,10,0,13" Width="280" Name="listBox1" HorizontalAlignment="Left"
VerticalAlignment="Top">
<!-- Khai báo tạo lập một chỉ mục con trong hộp danh mục với thuộc tính màu nền-->
<ListBoxItem Background="Beige">
<!-- Lồng vào chỉ mục này một StackPanel để có thể chứa nhiều hơn 1 phần tử UI
con theo chiều ngang-->
<StackPanel Orientation="Horizontal">
<!-- Khai báo tạo lập một ảnh đồ uống ở đầu mỗi chỉ mục-->
<Image Width="50" Height="50" Source="orange_juice.jpg"></Image>
<!-- Khai báo tạo lập một dòng chữ chỉ tên đồ uống-->
<TextBlock Margin="5" VerticalAlignment="Center" FontFamily="Times New Roman"
FontStyle="Italic" FontSize="18">Nước cam tươi</TextBlock>
</StackPanel>
</ListBoxItem>
<!-- Khai báo chỉ mục 2 tương tự như trên-->
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Width="50" Height="50" Source="kiwi_juice.jpg"></Image>
<TextBlock Margin="5" VerticalAlignment="Center" FontFamily="Times New Roman"
FontStyle="Italic" FontSize="18">Nước kiwi ép</TextBlock>
</StackPanel>
</ListBoxItem>
<!-- Khai báo chỉ mục 3 tương tự như trên-->
<ListBoxItem Background="Beige">
<StackPanel Orientation="Horizontal">
<Image Width="50" Height="50" Source="mango_juice.jpg"></Image>
<TextBlock Margin="5" VerticalAlignment="Center" FontFamily="Times New Roman"
FontStyle="Italic" FontSize="18">Nước soài ép</TextBlock>
</StackPanel>
</ListBoxItem>
<!-- Khai báo chỉ mục 4 tương tự như trên-->
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Width="50" Height="50" Source="milk.jpg"></Image>
<TextBlock Margin="5" VerticalAlignment="Center" FontFamily="Times New Roman"
FontStyle="Italic" FontSize="18">Sữa tươi tiệt trùng</TextBlock>
</StackPanel>
</ListBoxItem>
<!-- Khai báo chỉ mục 5 tương tự như trên-->
<ListBoxItem Background="Beige">
<StackPanel Orientation="Horizontal">
<Image Width="50" Height="50" Source="coffee.jpg"></Image>
<TextBlock Margin="5" VerticalAlignment="Center" FontFamily="Times New Roman"
FontStyle="Italic" FontSize="18">Cà phê Espresso</TextBlock>
</StackPanel>
</ListBoxItem>
</ListBox>
```

Như vậy, điểm mấu chốt để bổ sung thêm các thuộc tính giao diện như ảnh, text, checkbox,..., vào mỗi chỉ mục của hộp danh sách chính là việc kết hợp các phần tử UI riêng lẻ tương ứng vào cùng một phần tử Panel nằm trong khai báo chỉ mục. Trong trường hợp này, với mỗi khai báo chỉ mục `<ListBoxItem>` ta thêm vào một

`<StackPanel Orientation="Horizontal">` theo chiều ngang, trong đó, chứa một phần tử `<Image>` và 1 phần tử `<TextBlock>`. Nguồn dữ liệu ảnh được xác định qua thuộc tính `Source="<tên ảnh đã được bổ sung vào project>"`.

Kết quả của đoạn code được minh hoạ trong hình 9.26.



Hình 9.26 – Danh mục đồ uống có kèm ảnh minh hoạ xây dựng bằng WPF

### 5.3. Hộp mở rộng (Expander)

Hộp mở rộng Expander là một trong những điều khiển UI mới được giới thiệu trong WPF như một điều khiển cobả n. Expander cho phép thu gọn hoặc mở rộng một nội dung nào đó chứa trong nó, giống như một node trong TreeView, bằng việc click vào biểu tượng mũi tên (hướng lên, nếu điều khiển đang ở trạng thái mở rộng; hướng xuống, nếu đang ở trạng thái thu gọn). Điều khiển này rất tiện lợi: Khi diện tích form chính quá chật hẹp vì nhiều chức năng được trình bày trên cùng giao diện, ta có thể sử dụng Expander để chứa một số chức năng ít dùng có thể tạm thời được ẩn dưới một tên nhóm chung.

Trong ví dụ sau đây, ta sẽ làm một menu chứa 2 mục là đồ uống và đồ ăn, mỗi mục sẽ chứa danh sách các sản phẩm tương ứng mà nhà hàng cung cấp. Ta sử dụng Expander để có thể mở rộng/thu gọn từng mục nêu trên. Sau đây là mã XAML của ứng dụng:

```
<Window x:Class="Lesson3.Window2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Lesson 3 - Expander WPF Sample" Height="250" Width="200">
<!--Sử dụng stackpanel làm layout chính-->
<StackPanel>
<!--Khai báo tạo lập Expander chứa danh mục đồ uống-->
<Expander FontFamily="Times New Roman" FontSize="14" FontStyle="Oblique"
Header="Đồ uống" Background="#acb433" >
<!--Khai báo ListBox chứa danh mục đồ uống-->
<ListBox >
<!--Khai báo các chỉ mục con trong danh mục đồ uống-->
```

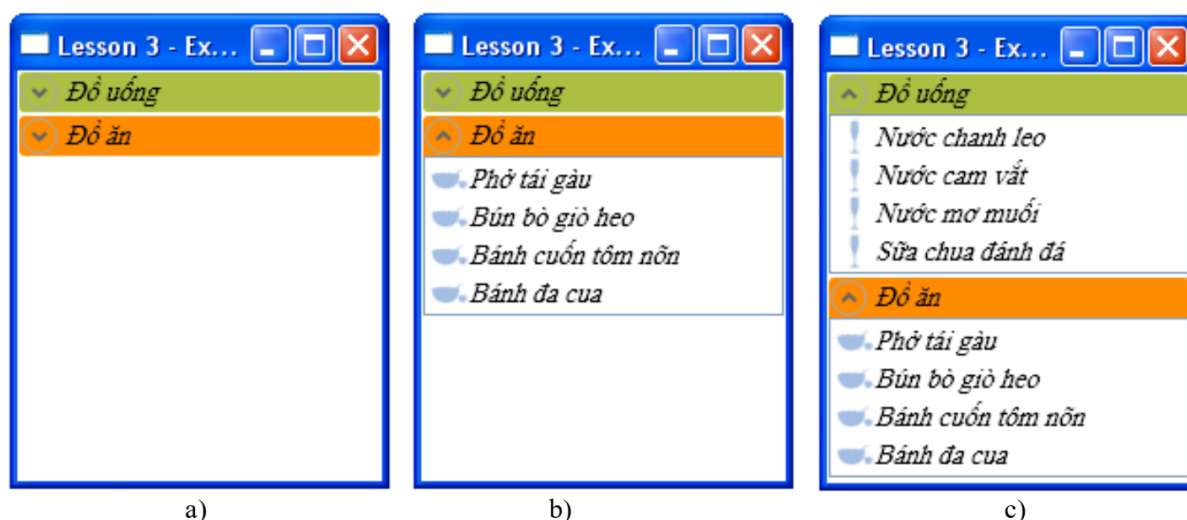


```

<!--Mỗi mục con được trang trí bằng một biểu tượng đi kèm text-->
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Source="glass.jpg" Height="20"></Image>
<TextBlock VerticalAlignment="Center">Nước chanh leo</TextBlock>
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Source="glass.jpg" Height="20"></Image>
<TextBlock VerticalAlignment="Center">Nước cam vắt</TextBlock>
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Source="glass.jpg" Height="20"></Image>
<TextBlock VerticalAlignment="Center">Nước mơ muối</TextBlock>
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Source="glass.jpg" Height="20"></Image>
<TextBlock VerticalAlignment="Center">Sữa chua đánh đá</TextBlock>
</StackPanel>
</ListBoxItem>
</ListBox>
</Expander>
<!--Khai báo Expander chứa danh mục đồ ăn-->
<Expander FontFamily="Times New Roman" FontSize="14" FontStyle="Oblique" Header="Đồ
ăn" Background="DarkOrange">
<ListBox>
<!--Khai báo các chỉ mục con trong danh mục đồ ăn-->
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Source="bowl.gif" Height="20"></Image>
<TextBlock VerticalAlignment="Center">Phở tái gà</TextBlock>
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Source="bowl.gif" Height="20"></Image>
<TextBlock VerticalAlignment="Center">Bún bò giò heo</TextBlock>
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Source="bowl.gif" Height="20"></Image>
<TextBlock VerticalAlignment="Center">Bánh cuốn tôm nõn</TextBlock>
</StackPanel>
</ListBoxItem>
<ListBoxItem>
<StackPanel Orientation="Horizontal">
<Image Source="bowl.gif" Height="20"></Image>
<TextBlock VerticalAlignment="Center">Bánh đa cua</TextBlock>
</StackPanel>
</ListBoxItem>
</ListBox>
</Expander>
</StackPanel>
</Window>

```

Như ta thấy việc sử dụng Expander trong WPF rất đơn giản, với cùng nguyên tắc như các điều khiển UI cơ bản khác. Ở đây ta sử dụng 2 hộp mở rộng Expander: một chứa danh mục đồ uống được đặt trong một ListBox; một chứa danh mục đồ ăn trong một ListBox. Kết quả của đoạn code được minh họa trong Hình 9.27.



Hình 9.27 – Tạo lập và sử dụng hộp mở rộng bằng WPF: a) Hai danh mục cùng thu gọn; b) Danh mục Đồ ăn được mở rộng; c) Cả hai danh mục được mở rộng (Danh mục trên đây danh mục dưới xuống)

## 5.4. Hộp soạn văn bản đa năng (RichTextBox)

Hộp soạn văn bản đa năng RichTextBox là một trong những điều khiển có chức năng phong phú. Không chỉ cho phép soạn sửa và hiển thị các nội dung text đơn thuần, RichTextBox còn cho phép thay đổi phông chữ (Verdana, Times New Roman,...), kiểu chữ (ngghiêng, đậm, gạch chân),... Đặc biệt, điều khiển RichTextBox trong WPF/.NET 3.0 còn cho phép kiểm tra/gợi ý sửa đổi lỗi chính tả tiếng Anh của nội dung văn bản chứa trong đó. RichTextBox trong WPF/.NET 3.0 là phần tử được cải tiến về cơ bản so với phiên bản trước của điều khiển RichTextBox trong .NET 2.0. Tuy nhiên, cùng với sự mở rộng về chức năng là việc bổ sung các API mới cũng như những cách thức sử dụng khác.

### 5.4.1 Chức năng cơ bản

Để thêm mới một hộp soạn thảo đa năng vào form, ta dùng mã XAML như sau:

```
<RichTextBox x:Name="XAMLRichBox" SpellCheck.IsEnabled="True"
MinHeight="100"></RichTextBox>
```

Thuộc tính **x:Name** là từ khoá xác định danh tính của RichTextBox được tạo. Thuộc tính này đóng vai trò là tham chiếu cho phép ta sau này buộc mã lệnh C# vào điều khiển. Thuộc tính **MinHeight** xác định số dòng có thể thấy được của hộp soạn thảo, giá trị này ngầm định bằng Thuộc tính **SpellCheck.IsEnabled="True"** kích hoạt tính năng kiểm tra lỗi chính tả tiếng Anh trong nội dung văn bản và gợi ý những từ đúng có thể để thay thế, giống như Microsoft Word. Tuy nhiên, nếu chỉ với một **RichTextBox**, ta không có cách nào để sửa đổi định dạng của văn bản trong **RichTextBox** như đánh chữ nghiêng, chữ đậm, đổi phông chữ, v.v. Muốn đạt được điều này, ta phải buộc mã lệnh vào giao diện Command của

## *RichTextBox.*

### 5.4.2 Giao diện Command

Microsoft chủ trương để người phát triển làm việc với RichTextBox thông qua giao diện Command. Mặc dù khái niệm này không mới đối với phần lớn người phát triển giao diện đồ họa người dùng, việc cài đặt và cú pháp trong XAML có chút khác biệt. Ta cần thêm một ToolBar và một số nút bấm hai trạng thái (ToggleButton) để gắn lệnh điều khiển RichTextBox đã tạo. Thuộc tính **Command** trên mỗi điều khiển kể trên sẽ xác định chức năng mà ta muốn kích hoạt trên RichTextBox,. Trong khi đó, thuộc tính **CommandTarget** xác định RichTextBox nào ta muốn chức năng kích hoạt của các nút bấm nhằm vào. Sau đây là đoạn mã XAML bổ sung thêm một ToolBar và 3 nút bấm hai trạng thái:

```
<!--Khai báo sử dụng Toolbar-->
<ToolBar>
<!--Khai báo nút bấm kích hoạt lệnh tô đậm đoạn chữ được chọn trong
RichTextBox-->
<ToggleButton MinWidth="40"
Command="EditingCommands.ToggleBold"
CommandTarget="{Binding ElementName=XAMLRichTextBox}"
TextBlock.FontWeight="Bold">B</ToggleButton>
<!--Khai báo nút bấm kích hoạt lệnh in nghiêng đoạn chữ được chọn trong
RichTextBox-->
<ToggleButton MinWidth="40"
Command="EditingCommands.ToggleItalic"
CommandTarget="{Binding ElementName=XAMLRichTextBox}"
TextBlock.FontStyle="Italic">I</ToggleButton>
<!--Khai báo nút bấm kích hoạt lệnh gạch chân đoạn chữ được chọn trong
RichTextBox-->
<ToggleButton MinWidth="40"
Command="EditingCommands.ToggleUnderline"
CommandTarget="{Binding ElementName=XAMLRichTextBox}">
<TextBlock TextDecorations="Underline">U</TextBlock>
</ToggleButton>
</ToolBar>
```

Mặc dù đoạn mã ví dụ chỉ bao gồm một số ít các nút lệnh

(Command="EditingCommands.ToggleBold",

Command="EditingCommands.ToggleBold", Command="EditingCommands.ToggleItalic"), có

tổng cộng 47 lệnh khác nhau mà ta có thể lựa chọn (có thể xem chúng bằng cách khảo sát

lớp **EditingCommands**).

Dưới đây là đoạn mã XAML đầy đủ cho phép ta xây dựng một hộp soạn thảo văn bản có thể thay đổi được kiểu chữ

(đậm, nghiêng, gạch chân):

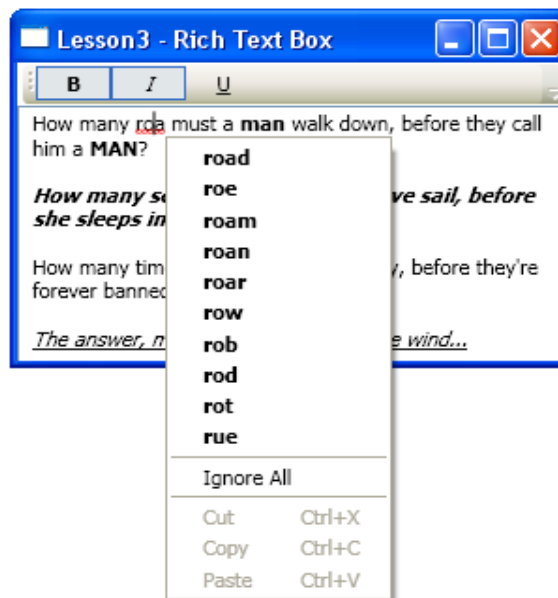
```
<Window x:Class="Lesson3.Window3"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Lesson3 - Rich Text Box" Height="300" Width="300"
>
<!--Sử dụng StackPanel làm layout chính-->
<StackPanel Orientation="Vertical">
<!--Khai báo toolbar-->
<ToolBar>
<!--Nút tô đậm-->
<ToggleButton MinWidth="40" Command="EditingCommands.ToggleBold"
CommandTarget="{Binding ElementName=XAMLRichTextBox}"
```

```

TextBlock.FontWeight="Bold">B
</ToggleButton>
<!--Nút in nghiêng-->
<ToggleButton MinWidth="40" Command="EditingCommands.ToggleItalic"
CommandTarget="{Binding ElementName=XAMLRichTextBox}"
TextBlock.FontStyle="Italic">I
</ToggleButton>
<!--Nút gạch chân-->
<ToggleButton MinWidth="40"
Command="EditingCommands.ToggleUnderline"
CommandTarget="{Binding ElementName=XAMLRichTextBox}">
<TextBlock TextDecorations="Underline">U</TextBlock>
</ToggleButton>
</ToolBar>
<!--Khai báo tạo lập RichTextBox-->
<RichTextBox x:Name="XAMLRichTextBox" SpellCheck.IsEnabled="True" MinHeight="100">
</RichTextBox>
</StackPanel>
</Window>

```

Kết quả được minh họa trong Hình 9.28.



Hình 9.28 – Xây dựng hộp soạn thảo đa năng đơn giản với các chức năng thay đổi kiểu chữ bằng WPF

## Tài liệu tham khảo

- [1]. Windows Presentation Foundation, URL: <http://msdn.microsoft.com/en-us/library/ms754130.aspx>.
- [2]. Introducing Windows Presentation Foundation, URL: <http://msdn.microsoft.com/en-us/library/aa663364.aspx>.
- [3]. WPF Architecture, URL: <http://msdn.microsoft.com/en-us/library/ms750441.aspx>.
- [4]. WPF Tutorial, URL: <http://dotnetslackers.com/articles/silverlight/WPFTutorial.aspx>.
- [5]. WPF Tutorials, URL: <http://www.wpfutorial.net/>.

[6]. Windows Presentation Foundation Control, <http://msdn.microsoft.com/en-us/library/ms752069.aspx>