Rozdział 15.
Przegląd pojemników WPF

Autor: Jacek Matulewski, jacek@fizyka.umk.pl

Materiały udostępnione słuchaczom Podyplomowego Studium Programowania i Zastosowań Komputerów, działającemu na Wydziale Fizyki, Astronomii i Informatyki Stosowanej Uniwersytetu Mikołaja Kopernika.

Do tej pory poznaliśmy trzy pojemniki (nazywane też mniej elegancko kontenerami) odpowiedzialne za układanie kontrolek w graficznym interfejsie użytkownika. Są to: `Grid`, `StackPanel` i `DockPanel`. W pojemniku `StackPanel` kontrolki ułożone są w poziomie lub pionie w jednej linii. W pojemniku `Grid` można zdefiniować siatkę komórek, w których rozmieszczamy kontrolki. Z kolei `DockPanel` pozwala na „przylepianie" kontrolek do krawędzi okna. To chyba trzy najczęściej wykorzystywane pojemniki.

Czym w ogóle są pojemniki? Są to elementy XAML, których klasy dziedziczą po klasie `Panel` z przestrzeni `System.Windows.Controls`, a które aranżują ułożenie kontrolek, znajdujących się w ich wnętrzu (dla których są rodzicami). W odróżnieniu od większości kontrolek, w pojemnikach można umieszczać wiele kontrolek. Temat rozdziału jest jednak szerszy: wspomnę bowiem także o innych kontrolach, które mogą przechowywać wiele elementów, mogą nimi być także kontrolki, takich jak wszelkiego typu listy, jak poznana w rozdziale 4. kontrolka `ListBox` , poznana w rozdziale 9. kontrolka `ComboBox` lub `TreeView` z rozdziału 8. Pojemniki w węższym znaczeniu, a więc np. `Grid`, `StackPanel` czy `DockPanel` różnią się od kontrolek `ListBox`, `ComboBox` lub `TreeView` tym, że nie dostarczają własnego widoku. Organizują jedynie położenie i wielkość innych kontrolek. Można jednak zmienić np. ich tło. Przycisk nie jest pojemnikiem, bo choć możemy w nim umieścić dowolną kontrolkę, to tylko jedną. Jeżeli chcemy ich więcej, to musimy użyć właśnie któregoś z pojemników.

Pojemniki (*Layout Containers*)

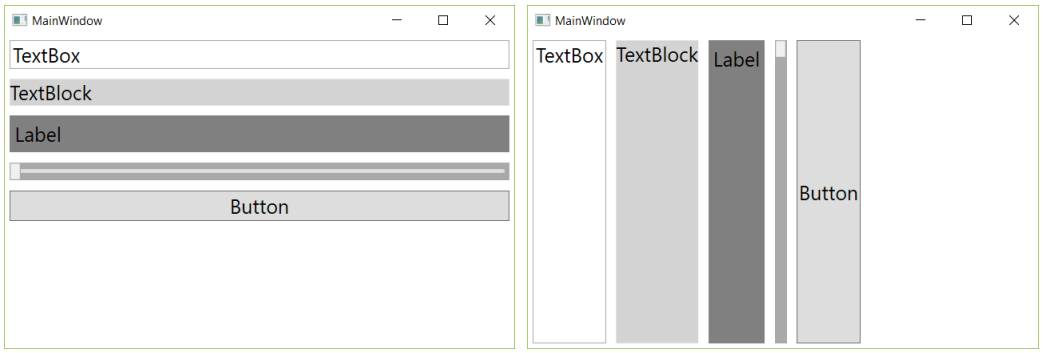
Poniżej przedstawię przegląd pojemników, w których będę prezentował zbiór kilku, zawsze tych samych kontrolek. Nie będzie w tym jakiejś większej filozofii. Chodzi mi przede wszystkim o to, żeby Czytelnik zobaczył co ma do dyspozycji. Nie będę przy tym zagnieżdżał kontrolek, żeby niepotrzebnie nie komplikować przeglądu, ale należy sobie zdawać sprawę, że jest to jak najbardziej możliwe.

Zacznijmy od czegoś, co już dobrze znamy, a więc od pojemnika `StackPanel`. Stwórzmy nową aplikację WPF i w pliku *MainWindow.xaml* umieśćmy kod z listingu 15.1. Zobaczymy kontrolki ułożone w jednej kolumnie (rysunek 15.1, lewy). Jak pamiętamy z rozdziału 10., możemy do elementu `StackPanel` dodać atrybut `Orientation` z wartością `Horizontal` i zmienić układ kontrolek na poziomy (rysunek 15.1, prawy).

Listing 15.1. Kilka kontrolek zorganizowanych przez pojemnik StackPanel. Dodałem kolory tła, aby lepiej widoczne były obszary kontrolek

```
<Window x:Class="PojemnikiWPF.MainWindow"
        ...
        Title="MainWindow" Height="350" Width="525"
        FontSize="20">
    <StackPanel>
        <TextBox Margin="5" Text="TextBox" />
```

```
            <TextBlock Margin="5" Text="TextBlock" Background="LightGray" />
            <Label Margin="5" Content="Label" Background="Gray" />
            <Slider Margin="5" Background="DarkGray" />
            <Button Margin="5" Content="Button" />
        </StackPanel>
    </Window>
```

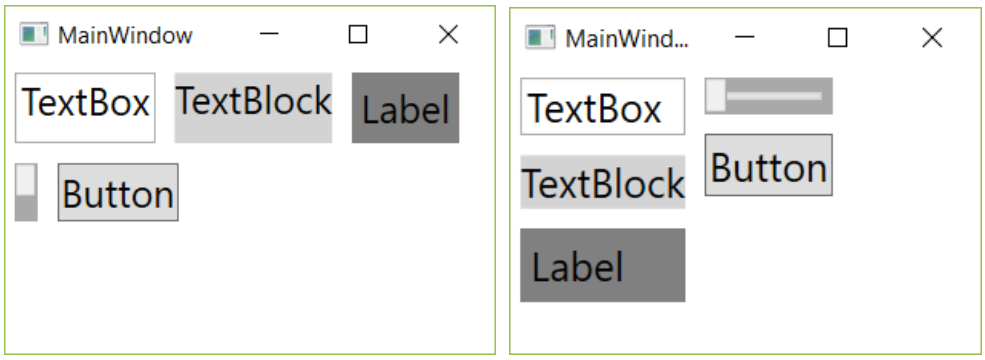


Rysunek 15. 1. Kontrolki, których ułożenie kontrolowane jest przez pojemnik StackPanel

Zmieńmy teraz typ pojemnika na `WrapPanel` nie zmieniając jego zawartości (listing 15.2). Kontrolki zmienią ułożenie: nadal będą układane w pionie lub poziomie (w tym pojemniku też działa atrybut `Orientation`), ale nie będą zajmować całej szerokości lub wysokości okna, a po dotarciu do krawędzi okna, będą układane w kolejnej kolumnie lub rzędzie (rysunek 15.2).

Listing 15.2. Zmiana pojemnika na WrapPanel

```
    <Window x:Class="PojemnikiWPF.MainWindow"
            ...
            Title="MainWindow" Height="350" Width="525"
            FontSize="20">
        <WrapPanel Orientation="Vertical">
            <TextBox Margin="5" Text="TextBox" />
            <TextBlock Margin="5" Text="TextBlock" Background="LightGray" />
            <Label Margin="5" Content="Label" Background="Gray" />
            <Slider Margin="5" Background="DarkGray" />
            <Button Margin="5" Content="Button" />
        </WrapPanel>
    </Window>
```



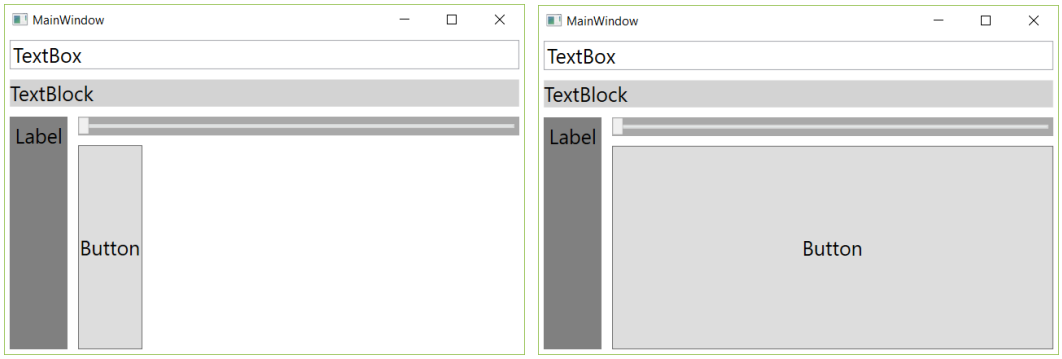
Rysunek 15.2. Jak wyżej, ale dla pojemnika Wrap Panel

Kolejny pojemnik, o którym należy wspomnieć to `DockPanel`. Nie ma on własności `Orientation`. Zamiast tego w każdej kontrolce-dziecku można użyć własności doczepianej `DockPanel.Dock`, która wskaże krawędź okna, do którego kontrolka ma być „przyklejona” (listing 15.3). Kolejność kontrolek z tą samą własnością w kodzie XAML wyznacza ich kolejność ustawienia od krawędzi (pierwsza jest najbliżej). Kolejność kontrolek w

kodzie XAML wyznacza również ich pierwszeństwo w rogach okna gdy „przyklejone” są do różnych krawędzi o wspólnym wierzchołku. Wśród możliwych wartości własności `DockPanel.Dock` nie ma takiej, która oznaczałaby wypełnienie (jak `Fill` w Windows Forms). Wolne miejsce pozostawione w pojemniku przez pozostałe kontrolki-dzieci może wypełnić tylko ostatnia kontrolka, jeżeli ustawimy własność pojemnika `LastChildFill` na `True` (rysunek 15.3, prawy).

Listing 15.3. Użycie pojemnika DockPanel

```
<Window x:Class="PojemnikiWPF.MainWindow"
        ...
        Title="MainWindow" Height="350" Width="525"
        FontSize="20">
    <DockPanel LastChildFill="False">
        <TextBox Margin="5" DockPanel.Dock="Top" Text="TextBox" />
        <TextBlock Margin="5" DockPanel.Dock="Top" Text="TextBlock"
                   Background="LightGray" />
        <Label Margin="5" DockPanel.Dock="Left" Content="Label" Background="Gray" />
        <Slider Margin="5" DockPanel.Dock="Top" Background="DarkGray" />
        <Button Margin="5" Content="Button" />
    </DockPanel>
</Window>
```

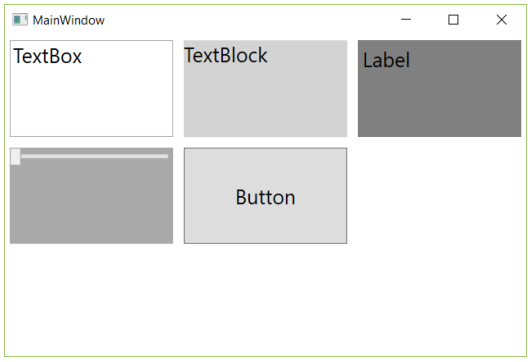


Rysunek 15.3 Jak wyżej, ale dla pojemnika DockPanel. Z prawej ułożenie w sytuacji, gdy atrybut LastChildFill ustawiony jest na True

Zmieńmy teraz pojemnik na `UniformGrid` (listing 15.4). To prosty w działaniu pojemnik, który dzieli okno na komórki o równej szerokości i wysokości. Automatycznie przypisuje poszczególne kontrolki do kolejnych komórek nadając im rozmiar wypełniający całą komórkę (rysunek 15.4). Przy zmianie rozmiaru okna, rozmiary komórek zmieniają się proporcjonalnie.

Listing 15.4. Pojemnik UniformGrid

```
<Window x:Class="PojemnikiWPF.MainWindow"
        ...
        Title="MainWindow" Height="350" Width="525"
        FontSize="20">
    <UniformGrid Columns="3" Rows="3">
        <TextBox Margin="5" Text="TextBox" />
        <TextBlock Margin="5" Text="TextBlock" Background="LightGray" />
        <Label Margin="5" Content="Label" Background="Gray" />
        <Slider Margin="5" Background="DarkGray" />
        <Button Margin="5" Content="Button" />
    </UniformGrid>
</Window>
```

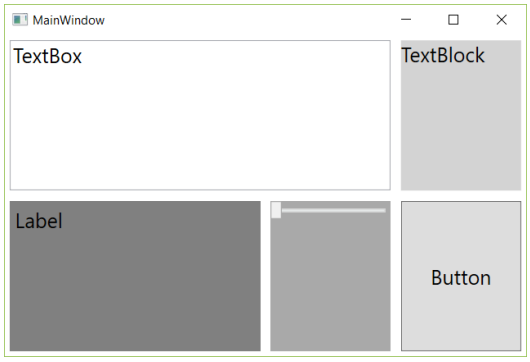


Rysunek 15.4. Jak wyżej, ale dla pojemnika UniformGrid

Gdy tworzymy nowy projekt, domyślnym pojemnikiem kontrolującym położenie kontrolek jest `Grid` (z ang. siatka). Nie będę o niej pisał zbyt wiele, bo używaliśmy jej dotąd już wiele razy. Siatka pozwala na dowolne ułożenie kontrolek w komórkach, na co, inaczej niż w pojemniku `UniformGrid`, mamy wpływ. Rozmiary komórek też możemy określać dzieląc siatkę na kolumny i wiersze o wskazanych szerokościach i wysokościach (własności `ColumnDefinitions` i `RowDefinitions`). Rozmiary wierszy możemy określać bezwzględnie podając liczbę pikseli lub względnie (zob. użycie gwiazdki w listingu 15.5). Kontrolki przypisujemy do komórek używając własności doczepianych `Grid.Column` i `Grid.Row` (pierwszy wiersz i pierwsza kolumna mają numer 0). Nie jesteśmy jednak całkowicie ograniczeni strukturą siatki. Możemy użyć własności doczepianych `Grid.ColumnSpan` i `Grid.RowSpan`, żeby rozciągnąć kontrolkę na dwie lub więcej kolumn i wierszy. Przykładem jest kontrolka `TextBox` rozciągnięta na dwie kolumny w listingu 15.5 (rysunek 15.5). Kontrolka nie musi też zajmować całej komórki.

Listing 15.5. Użycie siatki do ułożenia kontrolek

```
<Window x:Class="PojemnikiWPF.MainWindow"
        ...
        Title="MainWindow" Height="350" Width="525"
        FontSize="20">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBox Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2" Margin="5"
                 Text="TextBox" />
        <TextBlock Grid.Row="0" Grid.Column="2" Margin="5" Text="TextBlock"
                   Background="LightGray" />
        <Label Grid.Row="1" Grid.Column="0" Margin="5" Content="Label"
               Background="Gray" />
        <Slider Grid.Row="1" Grid.Column="1" Margin="5" Background="DarkGray" />
        <Button Grid.Row="1" Grid.Column="2" Margin="5" Content="Button" />
    </Grid>
</Window>
```



Rysunek 15.5. Jak wyżej, ale dla pojemnika Grid

Ostatnim pojemnikiem WPF jest `Canvas`, który słusznie kojarzony jest z grafiką. W tym pojemniku położenie kontrolek wyznaczane jest przez podanie odległości od dwóch wskazanych krawędzi okna (listing 15.6, rysunek 15.6). Pojemnik Canvas odtwarza bezwzględne pozycjonowanie kontrolek, jakie znamy z Windows Forms.

Listing 15.6. Pojemnik Canvas (z ang. płótno)

```
<Window x:Class="PojemnikiWPF.MainWindow"
        ...
        Title="MainWindow" Height="350" Width="525"
        FontSize="20">
    <Canvas>
        <TextBox Canvas.Left="0" Canvas.Top="0" Margin="5" Text="TextBox" />
        <TextBlock Canvas.Left="0" Canvas.Top="50" Margin="5" Text="TextBlock"
                   Background="LightGray" />
        <Label Canvas.Left="0" Canvas.Top="100" Margin="5" Content="Label"
               Background="Gray" />
        <Slider Canvas.Right="0" Canvas.Top="0" Width="200" Margin="5"
                Background="DarkGray" />
        <Button Canvas.Right="0" Canvas.Bottom="0" Margin="5" Content="Button" />
    </Canvas>
</Window>
```

Rysunek 15.6. Jak wyżej, ale dla pojemnika Canvas

Wszystkie dostępne w WPF pojemniki podsumowuje tabela 15.1. Warto jeszcze wspomnieć o wygodnym pojemniku, który pojawia się tylko w UWP. Jest nim `RelativePanel`. W pojemniku tym możemy wskazywać relatywne położenie elementów interfejsu użytkownika wskazując, że kontrolka ma leżeć pod, nad, z lewej lub z prawej strony innej kontrolki zidentyfikowanej poprzez nazwę. Dzięki temu powstaje elastyczny interfejs użytkownika, którym łatwiej zarządzać na różnych rozmiarach ekranu. Wsparciem w tym zakresie jest menedżer stanów wizualnych, w którym można określić ułożenie kontrolek dla różnych typów ekranu.

Tabela 15.1 Podsumowanie pojemników WPF

Klasa/Element	Własności	Własności doczepiane
`StackPanel`	`Orientation`	
`WrapPanel`	`Orientation`	
`DockPanel`	`LastChildFill`	`DockPanel.Dock`
`UniformGrid`	`Columns`, `Rows`	
`Grid`	`ColumnDefinitions`, `RowDefinitions`	`Grid.Column`, `Grid.Row`, `Grid.ColumnSpan`, `Grid.RowSpan`
`Canvas`		`Canvas.Left`, `Canvas.Top`, `Canvas.Right`, `Canvas.Bottom`

Kontrolki ułożenia (Layout Controls)

To nie koniec. Oprócz pojemników organizujących ułożenie kontrolek mamy jeszcze kilka kontrolek, które w różny sposób modyfikują wyświetlanie kontrolek lub ich grup. Wspominam tu o nich, bo są często używane w kontekście pojemników. Świetnym przykładem jest `ScrollViewer`, którego poznaliśmy już w rozdziale 2., a który pozwala na przewijanie umieszczonej w nim zawartości (jednej kontrolki lub pojemnika z wieloma kontrolkami). Jeżeli umieścimy w nim pole tekstowe, tak jak zrobiliśmy w rozdziale 2., to gdy rozmiar wyświetlanego tekstu będzie przekraczał rozmiar kontroli, ScrollViewer umożliwi jego przewijanie. Podobnie zadziała w przypadku listy. Jeżeli elementów np. w pojemniku `StackPanel` jest tak dużo, że „wystają" poza rozmiar kontrolki, `ScrollViewer` także umożliwi ich przewijanie.

Aby przetestować działanie tej kontrolki, wróćmy do pojemnika `StackPanel` z ułożeniem kontrolek w pionie (listing 15.1) na razie bez kontrolki `ScrollViewer`. Gdy po uruchomieniu aplikacji zmniejszymy wysokość okna, kontrolki zostaną zwyczajnie obcięte. Jednak jeżeli pojemnik umieścimy w elemencie `ScrollViewer`, to po zmniejszeniu wysokości okna, aktywny stanie pasek przewijania pionowego, który pozwoli nam przewinąć zawartość pojemnika tak, że możemy dostać się do wszystkich umieszczonych w nim kontrolek.

Listing 15.7. Użycie ScrollViewer

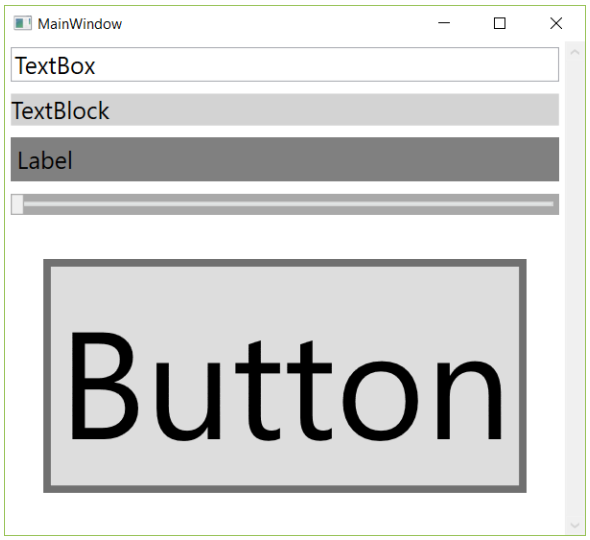
```
<Window x:Class="PojemnikiWPF.MainWindow"
        ...
        Title="MainWindow" Height="350" Width="525"
        FontSize="20">
    <ScrollViewer>
        <StackPanel ScrollViewer.VerticalScrollBarVisibility="Auto"
ScrollViewer.HorizontalScrollBarVisibility="Auto">
            <TextBox Margin="5" Text="TextBox" />
            <TextBlock Margin="5" Text="TextBlock" Background="LightGray" />
            <Label Margin="5" Content="Label" Background="Gray" />
            <Slider Margin="5" Background="DarkGray" />
            <Button Margin="5" Content="Button" />
        </StackPanel>
    </ScrollViewer>
</Window>
```

Inną ciekawą kontrolką modyfikującą sposób wyświetlania innych kontrolek jest `ViewBox`. Dla przykładu użyjmy jej na kontrolce `Button` (listing 15.8). Powoduje ona, że kontrolka umieszczona wewnątrz niej stara się zająć całą dostępną przestrzeń. Nie oznacza to jednak prostej zmiany jej rozmiaru. Na kontrolce wewnątrz `ViewBox` wykonywana jest bowiem transformacja skalowania od domyślnych rozmiarów do rozmiaru, który

wypełni całą szerokość pojemnika `StackPanel`. Dzięki skalowaniu, powiększona będzie nie tylko sama kontrolka, ale także jej zawartość – w naszym przykładzie jest nią kontrolka `TextBlock` wyświetlająca etykietę w przycisku. Przeskalowane zostaną także marginesy przycisku. Pokazuje to rysunek 15.7.

Listing 15.8. Użycie ViewBox

```
<Window x:Class="PojemnikiWPF.MainWindow"
        ...
        Title="MainWindow" Height="350" Width="525"
        FontSize="20">
    <ScrollViewer>
        <StackPanel ScrollViewer.VerticalScrollBarVisibility="Auto"
                    ScrollViewer.HorizontalScrollBarVisibility="Auto">
            <TextBox Margin="5" Text="TextBox" />
            <TextBlock Margin="5" Text="TextBlock" Background="LightGray" />
            <Label Margin="5" Content="Label" Background="Gray" />
            <Slider Margin="5" Background="DarkGray" />
            <Viewbox Stretch="Uniform">
                <Button Margin="5" Content="Button" />
            </Viewbox>
        </StackPanel>
    </ScrollViewer>
</Window>
```



Rysunek 15.7. Efekt użycia kontrolki ViewBox

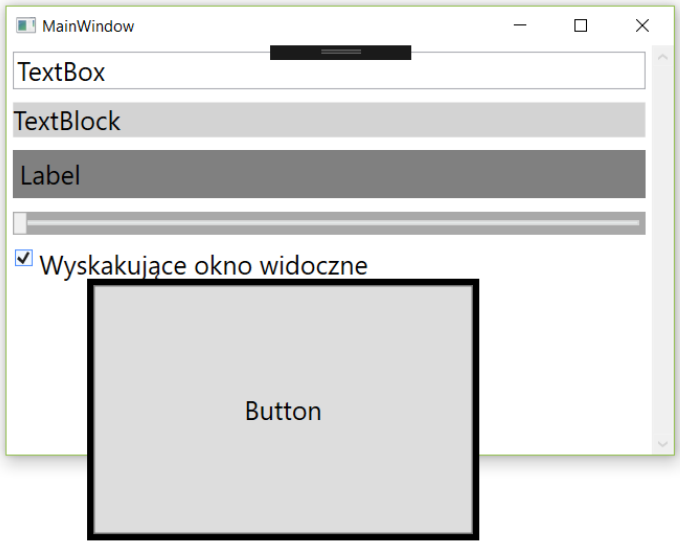
Kolejną ciekawą kontrolką jest `Popup`. „Wyjmuje” ona umieszczoną w niej zawartość z okna i umieszcza w dodatkowym wyskakującym okienku. Możemy określić jego wielkość i położenie na ekranie (np. względem położenia myszki). To dodatkowe okno jest widoczne lub ukryte w zależności od wartości własności `IsOpen` (rysunek 15.8). W przykładzie widocznym na listingu 15.9. związaliśmy tą własność z własnością `IsChecked` kontrolki pola opcji (`CheckBox`)[1].

Listing 15.9. Użycie Popup

```
<Window x:Class="PojemnikiWPF.MainWindow"
```

[1] Wiązania będą omówione w rozdziale 18. O kontrolce `Popup` więcej na stronie *http:// www.c-sharpcorner.com/UploadFile/mahesh/using-xaml-popup-in-wpf/*

```
            ...
            Title="MainWindow" Height="350" Width="525"
            FontSize="20">
        <ScrollViewer>
            <StackPanel ScrollViewer.VerticalScrollBarVisibility="Auto"
    ScrollViewer.HorizontalScrollBarVisibility="Auto">
                <TextBox Margin="5" Text="TextBox" />
                <TextBlock Margin="5" Text="TextBlock" Background="LightGray" />
                <Label Margin="5" Content="Label" Background="Gray" />
                <Slider Margin="5" Background="DarkGray" />
                <CheckBox x:Name="checkBox" Margin="5" Content="Wyskakujące okno widoczne"
                          IsChecked="False" />
                <Popup Width="300" Height="200"
                       IsOpen="{Binding ElementName=checkBox, Path=IsChecked}"
                       Placement="MousePoint" HorizontalOffset="50" VerticalOffset="20">
                    <Button Margin="5" Content="Button" />
                </Popup>
            </StackPanel>
        </ScrollViewer>
    </Window>
```



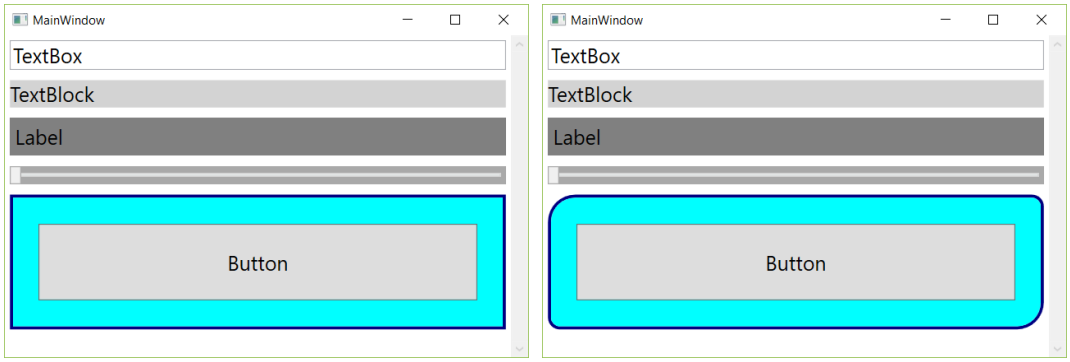
Rysunek 15.8. Efekt użycia kontrolki Popup

Na koniec warto też wspomnieć o prostej, ale bardzo użytecznej kontrolce `Border`, która dodaje obramowanie o podanej grubości i kolorze. Przykład widoczny jest na listingu 15.10. Do czterech wierzchołków dorysowywanego brzegu można dodać zaokrąglenia. Ich promienie ustalamy za pomocą atrybutu `CornerRadius` (rysunek 15.10, prawy).

Listing 15.10. Dodanie zewnętrznych krawędzi kontrolki za pomocą Border

```
    <Window x:Class="PojemnikiWPF.MainWindow"
            ...
            Title="MainWindow" Height="350" Width="525"
            FontSize="20">
        <ScrollViewer>
            <StackPanel ScrollViewer.VerticalScrollBarVisibility="Auto"
                        ScrollViewer.HorizontalScrollBarVisibility="Auto">
```

```
            <TextBox Margin="5" Text="TextBox" />
            <TextBlock Margin="5" Text="TextBlock" Background="LightGray" />
            <Label Margin="5" Content="Label" Background="Gray" />
            <Slider Margin="5" Background="DarkGray" />
            <Border BorderBrush="Navy" BorderThickness="3" Background="Cyan"
                    CornerRadius="25,10,25,10">
                <Button Margin="25" Height="100" Content="Button" />
            </Border>
        </StackPanel>
    </ScrollViewer>
</Window>
```



Rysunek 15.9. Efekt użycia kontrolki Border. Z prawej strony zdefiniowany został atrybut CornerRadius

Listy (*Items Controls*)

Wspomniałem, że nie tylko pojemniki mogą przechowywać wiele elementów. To potrafią też różnego typu listy (`ListBox`, `ListView`), drzewa (`TreeView`) i rozwijane listy (`ComboBox`). Zwykle pokazywane są w nich kolekcje łańcuchów, jednak tak naprawdę mogą przechowywać dowolne kontrolki WPF. I wcale nie są tak różne od pojemników omówionych wyżej (tj. klas dziedziczących po `Panel`). Dla przykładu kontrolka `ListBox` ma bardzo podobne działanie jak `StackPanel` w otoczeniu `ScrollViewer` (listing 15.11, rysunek 15.10). Nie warto jednak ponownie omawiać tej kontrolki. Używaliśmy jej już w rozdziale 4., a więcej jej możliwości, w szczególności ustalanie szablonu dla elementów, będzie przedstawionych w rozdziale ****.

Listing 15.11. Kontrolka ListBox

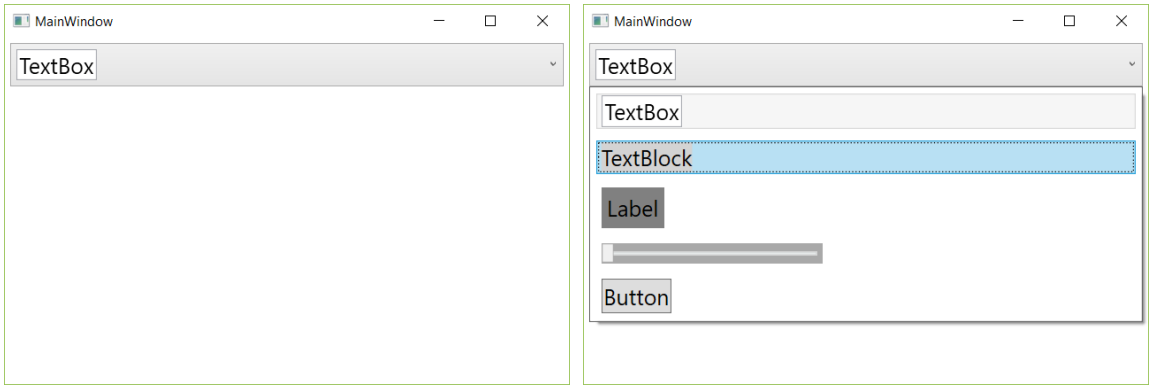
```
<Window x:Class="PojemnikiWPF.MainWindow"
        ...
        Title="MainWindow" Height="350" Width="525"
        FontSize="20">
    <StackPanel>
        <ListBox Margin="5" Padding="5">
            <ListBox.ItemContainerStyle>
                <Style TargetType="ListBoxItem">
                    <Setter Property="Margin" Value="5" />
                </Style>
            </ListBox.ItemContainerStyle>
            <TextBox Text="TextBox" />
```

```
            <TextBlock Text="TextBlock" Background="LightGray" />
            <Label Content="Label" Background="Gray" />
            <Slider Width="200" Background="DarkGray" />
            <Button Content="Button" />
        </ListBox>
    </StackPanel>
</Window>
```

Rysunek 15.10. Kontrolka ListBox

Poza `ListBox`, w WPF są jeszcze dwie inne podobne kontrolki `ListView` i `ItemsControl`. Wszystkie mogą wyświetlać listy elementów (kontrolek), ale w `ItemsControl` nie możemy zaznaczać elementów z listy. Wystarczy w kodzie z listingu 15.11 zmienić element `ListBox` na `ItemsControl` i uruchomić aplikację, aby się o tym przekonać. Klasa `ListBox` dziedziczy po `ItemsControl` dodając do niej właśnie możliwość zaznaczania w liście elementów (a przy okazji też paski przewijania). Z kolei `ListView` dziedziczy po `ListBox` dodając możliwość zmieniania widoków tj. sposobów w jaki wyświetlane są elementy.

Ostatnią kontrolką, jaką chcę tu zaprezentować jest `ComboBox`. Jedyna zmiana w kodzie z listingu 15.11 będzie polegała na zamianie `ListBox` na `ComboBox`. I w tej kontrolce możliwe jest zaznaczanie elementów – są one wówczas wyświetlane w jej podstawowym „zwiniętym” widoku (rysunek 15.11).



Rysunek 15.11. Kontrolka ComboBox, czyli rozwijana lista