ANNA KEMPA

# WPROWADZENIE

*do*


Tworzenie aplikacji w WPF przy użyciu

## XAML i C#

Helion

Autorka: Anna Kempa – pracownik naukowo-dydaktyczny Wydziału Informatyki i Komunikacji Uniwersytetu Ekonomicznego w Katowicach



Redaktor prowadzący: Małgorzata Kulik
Projekt okładki: Jan Paluch

Fotografia na okładce została wykorzystana za zgodą Shutterstock.com

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
*http://helion.pl/user/opinie/jchata*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:
*ftp://ftp.helion.pl/przyklady/jchata.zip*

ISBN: 978-83-283-3272-0



Printed in Poland.

# Spis treści

# Wstęp

**Windows Presentation Foundation** (WPF) firmy Microsoft jest jedną z wiodących technologii do tworzenia desktopowych aplikacji dla systemu Windows. Integruje interfejs użytkownika, grafikę 2D i 3D, multimedia oraz dokumenty. Umożliwia definiowanie interfejsu użytkownika w deklaratywnym języku XAML, a także pozwala na łatwą implementację wzorców projektowych, które oddzielają warstwę logiczną od warstwy prezentacji. Ponieważ jest zbudowany na bazie Direct3D, aplikacje WPF korzystają z przyspieszenia sprzętowego. Znaczącym walorem WPF jest możliwość kompozycji i adaptacji poszczególnych elementów, z których budowany jest interfejs.

Czy nauka tak zaawansowanego narzędzia jest trudna? Trud wkładany w naukę programowania (i wielu innych dziedzin) jest zazwyczaj stanem przejściowym, postrzeganym jako doraźne, tymczasowe problemy, które mogą wystąpić na każdym etapie nauki. Wraz z nabywaniem doświadczenia każdy programista uświadamia sobie, że ewentualne trudności tkwią nie tyle w stopniu skomplikowania danego narzędzia, ile w dostępności odpowiednich materiałów, w tym dokumentacji i podręczników. Starsze pokolenie programistów — które uczyło się programować jeszcze przed upowszechnieniem internetu i przy ograniczonej ofercie podręczników w księgarniach — szczególnie docenia ten aspekt.

Obecnie nie brakuje pomocy naukowych, a zróżnicowanie oferty pozwala na bardziej swobodny dobór źródeł dostosowanych do oczekiwań Czytelnika. Niniejszy podręcznik dedykowany jest osobom początkującym. Nie przedstawia wszystkich bazowych funkcjonalności dostarczanych przez WPF. Główny akcent położony jest na podstawach, umożliwiających zbudowanie biznesowej aplikacji, bez animacji i grafiki 3D. Kolejną cechą książki jest jej zadaniowy charakter — większość rozdziałów opisuje wykonanie konkretnej aplikacji.

# Dla kogo jest ta książka?

Podręcznik jest przeznaczony dla osób początkujących, które niedawno rozpoczęły naukę programowania i znają podstawy C#. Nie wymaga znajomości Windows Forms ani innych rozwiązań służących do tworzenia interfejsu graficznego. Celem książki

jest ułatwienie pierwszych kroków w zakresie technologii WPF i języka XAML. Takie przygotowanie umożliwi Czytelnikowi samodzielne wykonanie prostych aplikacji biznesowych, a ponadto utoruje drogę do dalszej nauki WPF w oparciu o bardziej zawansowane źródła, w tym dokumentację techniczną.

Wprowadzający charakter podręcznika pozwala polecać jego lekturę studentom i uczniom szkół średnich oraz wszystkim innym osobom, które mają powody i ochotę nauczyć się WPF i nie mają dużego doświadczenia informatycznego.

# Jak czytać tę książkę?

Podręcznik został przygotowany raczej do *pracy z WPF* niż do *czytania o WPF*. Rozdział 1. wprowadza w podstawowe zagadnienia dotyczące WPF i XAML, tak aby możliwie szybko można było przejść do praktycznego etapu nauki. Większość rozdziałów opisuje wykonanie kompletnych i niezależnych aplikacji.

Tytułowe pytanie powinno zatem brzmieć: jak pracować z tą książką? Zalecane jest wykonywanie omawianych programów według podanych objaśnień. Kody programów można pobrać ze strony *http://helion.pl/pobierz-przyklady/jchata/*. Dostępne są w dwóch wersjach: do nauki oraz w postaci gotowych projektów[1]. Polecam korzystanie przede wszystkim z tej pierwszej wersji, przeznaczonej do nauki, która zawiera pliki tekstowe z fragmentami kodu opatrzone krótkimi komentarzami. Na podstawie wyjaśnień ujętych w treści podręcznika można z owych fragmentów, niczym z klocków, budować program. Jest to sposób znany z wielu poradników i podręczników software'owych, który dobrze sprawdza się w przypadku nauki takich technologii jak WPF. Budowanie aplikacji z przygotowanych „klocków" nie polega jednak na bezmyślnym używaniu opcji *Kopiuj* i *Wklej*. Praca taka wymaga analizy poszczególnych fragmentów kodu w oparciu o wyjaśnienia zawarte w podręczniku i w innych źródłach (zwłaszcza jeśli ktoś uczy się jednocześnie C#). Przesadne dążenie do zrozumienia od razu wszystkiego w każdym fragmencie programu też nie jest wskazane. Wiele rzeczy będzie się powtarzać w kolejnych programach, w nowych odsłonach. Ponadto będą zadania do samodzielnego wykonania, różne modyfikacje, będzie wiele okazji do tego, aby przekonać się, czy dany mechanizm bądź konstrukcja są zrozumiałe. Jak wspomniałam kody programów dostępne są także w postaci gotowych projektów, do których można zajrzeć w przypadku większych trudności z uruchomieniem swojej wersji programu.

Na końcu większości rozdziałów znajdują się podrozdziały z zadaniami i wskazówkami. Czytelnik może samodzielnie pracować nad danym programem i w razie konieczności korzystać ze szczegółowych wskazówek i propozycji rozwiązań. Zachęcam Czytelnika do przeglądania dokumentacji technicznej w trakcie tych prac w celu bliższego poznania danej klasy, jej właściwości czy metod.

[1] Wśród gotowych projektów nie ma programów dla zadań do samodzielnego wykonania. Dla takich zadań umieszczono w treści podręcznika jedynie wskazówki i szczegółowe wyjaśnienia.

Zadaniowy charakter książki daje Czytelnikowi w trakcie pracy dużo okazji do satysfakcji. Tworzone programy prezentują wizualne, od razu widoczne efekty. Zachęcam to kreatywności i doskonalenia danego programu według własnych upodobań z użyciem poznanych na danym etapie konstrukcji i mechanizmów. Opisy objaśniające wykonanie danego programu, gdyby patrzeć na nie „z boku", to znaczy nie angażując się w proces tworzenia, mogą się wydać zbyt techniczne. Wynika to w dużej mierze ze specyfiki WPF (i innych zaawansowanych rozwiązań wspomagających tworzenie GUI), dla której trafne jest polskie przysłowie „diabeł tkwi w szczegółach". Tworzenie interfejsu wymaga ustalenia wielu drobnych detali, których opisanie nie brzmi jak lektura do poczytania. Ponadto WPF ma trochę swoich „osobliwości", do których należą m.in. właściwości dołączone, nazywane przez niektórych „magią WPF". Proszę się nie obawiać — to wszystko ma swoje solidne uzasadnienie i ani się Czytelnik zorientuje, jak zacznie swobodnie używać tych i innych poznanych tu rozwiązań WPF. Podsumowując: powody do zadowolenia znajdzie Czytelnik przede wszystkim w wyniku naszej współpracy — w postaci działających programów uruchamianych na swoim komputerze, a następnie samodzielnie modyfikowanych.

# Zakres książki

W WPF wiele rzeczy można zrobić na kilka różnych sposobów. W tym podręczniku przedstawiam tylko wybrane sposoby. Wśród zastosowanych kryteriów wyboru można wymienić popularność i jakość danego rozwiązania. Niemniej w niektórych przypadkach, zwłaszcza na początku książki, uwzględniam stopień trudności, prezentując prostszy sposób. To ostatnie kryterium traci na znaczeniu wraz z zaawansowaniem nauki w kolejnych rozdziałach. Zaraz po zapoznaniu się z podstawami WPF opisanymi w rozdziale 1. można przystąpić w rozdziale 2. do wykonania pierwszej prostej aplikacji, zawierającej dwa przyciski. Pierwsze programy będą tworzone w tak zwanym widoku autonomicznym (zbliżonym nieco do Windows Forms). Takie programowanie ma swoje wady, ujawniające się w bardziej złożonych projektach, do których zalicza się brak dostatecznej elastyczności czy utrudnienia dotyczące testowania aplikacji. Zaletą tego podejścia jest natomiast większa przystępność dla osób uczących się. Można wykorzystać tę zaletę i jeszcze przed wprowadzeniem bardziej zaawansowanych zagadnień omówić podstawowe elementy WPF, które są niezależne od stylu programowania. Program realizowany w rozdziale 3. będzie wymagał zdefiniowania większej liczby kontrolek, co zrodzi potrzebę bardziej uporządkowanego podejścia do planowania interfejsu. Odpowiedź na to zapotrzebowanie przynosi rozdział 4., opisujący standardowe panele, pozwalające zarządzać układem graficznym elementów. Zaraz później, w rozdziale 5., Czytelnik dokona kolejnego ważnego kroku, mianowicie pozna wiązanie danych. Wówczas okaże się, że część mrówczej pracy wykonanej w rozdziale 3. może być zastąpiona przez ten nieoceniony mechanizm WPF. W dwóch kolejnych rozdziałach zostały przedstawione aplikacje wykorzystujące kontrolki przewidziane dla kolekcji danych — `ListView` i `DataGrid`. Rozdziały 8. i 9. prezentują możliwości kontrolek służących do tworzenia menu oraz zakładek (`Menu` i `TabControl`) na przykładzie popularnych aplikacji — przeglądarki www i odtwarzacza audio. Rozdział 10. traktuje o ważnych zagadnieniach, takich jak zasoby, style i wyzwalacze, dzięki którym definiowanie interfejsu jest bardziej wygodne i profesjonalne. Bogate możliwości WPF w zakresie

kompozycji elementów interfejsu będzie można szerzej poznać w rozdziale 11., poświęconym przede wszystkim tematyce szablonów. Gdyby nie wiązanie danych, czyli specyficzny mechanizm WPF umożliwiający automatyczne aktualizacje danych, podręcznik mógłby się w tym miejscu już zakończyć. Ale korzystanie w pełni z komfortu, jaki zapewnia to rozwiązanie, wymaga poznania sposobów wykonywania konwersji danych oraz ich walidacji. Konwertery zostały omówione w jednym z podrozdziałów rozdziału 11., natomiast walidacja danych — w rozdziale 12. Rozdział kolejny zawiera wprowadzenie do wzorca projektowego MVVM, który opiera się na mechanizmie wiązania danych i tak zwanych powiadomieniach. Zasady tego wzorca odbiegają od podejścia z zastosowaniem widoku autonomicznego. Wzorzec MVVM pozwala na bardziej konsekwentną separację logiki aplikacji i sposobu wyświetlenia danych, co wpływa na zwiększenie elastyczności programu i ułatwia jego testowanie. Rozdział 14., ostatni, zawiera uporządkowanie pewnych aspektów teoretycznych, głównie poprzez omówienie hierarchii klas WPF. Spoglądanie na diagramy dziedziczenia klas niczym na mapy pozwoli Czytelnikowi wyłonić to, co poznał w tym podręczniku podczas wykonywania zadań, oraz to, co pozostało mu do poznania.

Podsumowując, zakres tej książki obejmuje: podstawy języka XAML (wykorzystywanego w WPF do deklaratywnego opisania interfejsu), większość kontrolek (elementów wywodzących się z klasy `Control`) oraz wybrane elementy wywodzące się z klasy `FrameworkElement`. Czytelnik tej książki łatwo przyswoi mechanizmy i rozwiązania dostarczane przez WPF, takie jak wiązanie danych, wyzwalacze, konwertery, zasoby, style i szablony. Pozna także wybrane sposoby walidacji danych i podstawy wzorca projektowego MVVM. To wszystko, po dodaniu umiejętności w zakresie baz danych (będących poza tematem tego podręcznika), powinno przygotować Czytelnika do napisania aplikacji biznesowej.

Gdyby porównać naukę WPF do zwiedzania jakiegoś większego miasta, to moglibyśmy powiedzieć, że ten podręcznik (jako przewodnik) proponuje uproszczoną trasę wycieczki po ciekawszych zakątkach miasta z pomięciem kilku ważnych miejsc, które poleca podróżnikowi w przyszłości (w tym grafika 3D i animacje), oraz z pominięciem wielu mniejszych zakątków. Podręcznik ten należy zatem traktować jako pierwszą podróż i przetarcie szlaków. Mam nadzieję, że z jego pomocą będzie Czytelnikowi łatwiej zgłębiać zaawansowane możliwości WPF podczas dalszej nauki.

Rozdział 5.

# Wiązanie danych — aplikacja Produkt

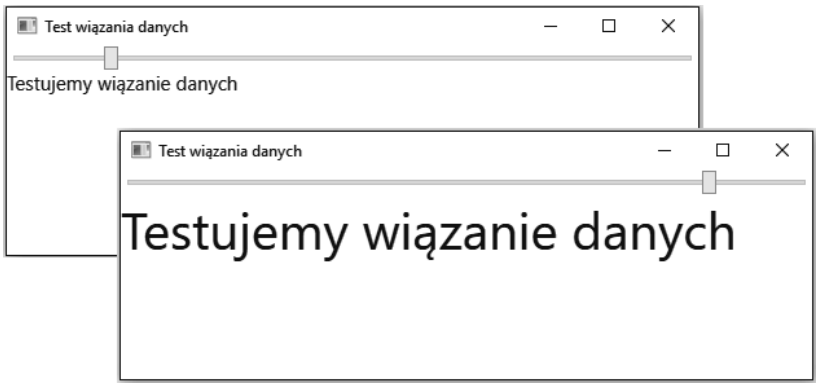
Wiązanie danych w WPF jest mechanizmem, który ustanawia połączenie między dwiema właściwościami różnych obiektów. Jednym z ważniejszych zastosowań tego mechanizmu jest łączenie danych i wizualnych elementów interfejsu. Definiując wiązanie, należy określić źródło danych i cel wiązania. Źródłem danych może być obiekt, baza danych, zasób, XML — właściwie „wszystko". Natomiast celem powinna być właściwość zależna elementu WPF. Aby móc w pełni korzystać z automatyzmu, jaki dostarcza ten mechanizm, należy jeszcze uwzględnić kilka innych rozwiązań WPF, takich jak konwertery, walidacja danych, powiadomienia o zmianach czy wiązanie poleceń. Zagadnienia te zostaną przedstawione w dalszej części książki. Niemniej już teraz będzie można docenić wygodę wiązania danych. W niniejszym rozdziale w pierwszej kolejności przetestujemy wiązanie danych na prostym przykładzie z suwakiem, a następnie przystąpimy do wykonania aplikacji, w której zostanie ustanowione połączenie między polami tekstowymi a właściwościami obiektu klasy `Produkt`. Wykorzystamy umiejętności nabyte w poprzednim rozdziale i umieścimy kontrolki w oknie przy użyciu panelu.

## 5.1 Testowanie wiązania danych

Wiązanie danych można wykonać w *code-behind* lub kodzie XAML. W tym przykładzie wykorzystamy ten drugi sposób. Otwórz nowy projekt WPF. W kodzie XAML ustaw atrybut `Title=" Test wiązania danych"`. Zmień także rozmiar okna aplikacji: `Height="200" Width="550"`. W oknie *XAML* podmień kod ze znacznikami `<Grid>` na następujący:

```
<StackPanel>
    <Slider x:Name="rozmiarTekstu" Minimum="10" Value="15" Maximum="45"/>
    <TextBlock FontSize="{Binding Path=Value, ElementName=rozmiarTekstu}">
        Testujemy wiązanie danych
    </TextBlock>
</StackPanel>
```

Programik ten można już uruchomić. Dzięki wiązaniu danych nie potrzebujemy obsługiwać zdarzenia dla zmiany wartości suwaka. Po uruchomieniu programu możemy przesuwać suwak i w ten sposób zwiększać lub zmniejszać rozmiar czcionki tekstu, jaki się wyświetla poniżej suwaka (rysunek 5.1).




**Rysunek 5.1.** *Testowanie wiązania danych z użyciem suwaka*

Suwak stanowi źródło dla omawianego wiązania danych. Natomiast blok tekstu jest celem tego wiązania. Zatem po obu stronach wiązania w tym przypadku są jakieś elementy WPF.

Omówmy dokładnie przedstawiony kod XAML. W znaczniku `Slider` zdefiniowany jest suwak. Kontrolka ta ma więcej właściwości, tu zostały wykorzystane następujące: wartość minimalna (`Minimum`), wartość aktualna (`Value`) oraz wartość maksymalna (`Maximum`).

Znacznik `TextBlock` definiuje blok tekstu. Bez wiązania danych definicja tego elementu mogłaby wyglądać przykładowo tak:

```
<TextBlock FontSize="15">
    Testujemy wiązanie danych
</TextBlock>
```

Albo w zapisie równoważnym tak:

```
<TextBlock FontSize="15" Text="Testujemy wiązanie danych"/>
```

Skupmy się zatem na tym, co różni ten prosty zapis, podobny do wielu innych prezentowanych wcześniej w tej książce, od zapisu uwzględniającego wiązanie. Spójrzmy na kod:

```
<TextBlock FontSize="{Binding Path=Value, ElementName=rozmiarTekstu}">
```

Zamiast konkretnej wartości dla atrybutu `FontSize` przypisane jest wyrażenie w klamrach. Wiązanie dwóch elementów w kodzie XAML realizowane jest za pomocą specjalnego rozszerzenia znaczników `Binding`. Definiując wiązanie, po lewej stronie znaku przypisania określamy cel wiązania, tu jest to właściwość `FontSize` (rozmiar tekstu). Natomiast po prawej stronie znaku przypisania umieszcza się źródło wiązania. W klamrach oprócz słowa `Binding` widzimy dwie właściwości: `ElementName`, która wskazuje źródło wiązania, oraz `Path`, która wskazuje właściwość obiektu źródłowego.

Warto nadmienić, że można użyć alternatywnego rozszerzenia znaczników poprzez przekazanie obiektu `Path` do konstruktora. Dzięki temu znacznik otwierający `TextBlock` możemy zapisać także w następujący sposób:

```
<TextBlock FontSize="{Binding Value, ElementName=rozmiarTekstu}">
```

Jak widać, nie ma tu jawnego przypisania wartości do właściwości `Path`.

Można także inaczej wykonać samo wiązanie w kodzie XAML — używając `Binding` jako elementu XAML, jak w przykładzie:

```
<TextBlock>
    <TextBlock.FontSize>
        <Binding Path="Value" ElementName="rozmiarTekstu"/>
    </TextBlock.FontSize>
    Testujemy wiązanie danych
</TextBlock>
```

Po „rozgrzewce" z suwakiem, która pozwoliła Czytelnikowi poznać istotę mechanizmu wiązania danych, przejdziemy w dalszej części rozdziału do pracy nad tworzeniem aplikacji *Produkt*. Zaczniemy od kodu XAML.

# 5.2 Kod XAML

Nierzadko mamy do czynienia z przypadkiem, w którym kilka różnych kontrolek jest wiązanych z tym samym obiektem źródłowym (ale z różnymi jego właściwościami). W takich sytuacjach można skorzystać z **kontekstu danych**, który umożliwia podanie źródła danych w jednym miejscu, w elemencie nadrzędnym. Definicja kontekstu wykonywana za pomocą właściwości `DataContext` może być umieszczona w kodzie XAML lub *code-behind*. W tym i w dwóch kolejnych rozdziałach użyjemy tego drugiego sposobu.

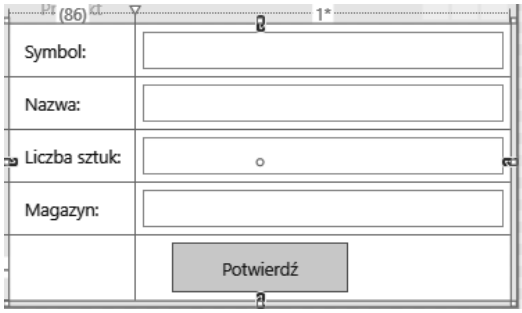
Otwórz nowy projekt WPF. W kodzie XAML ustaw atrybut `Title="Produkt"`. Zmień także rozmiar okna aplikacji: `Height="220" Width="350"`. W oknie *XAML* podmień kod ze znacznikami `<Grid>` i `</Grid>` na następujący:

```
<Grid x:Name="gridProdukt">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
```

```
        <Label Content="Symbol:" Grid.Row="0" Grid.Column="0" Margin="5"/>
        <TextBox Grid.Row="0" Grid.Column="1" Margin="5" Text="{Binding Symbol}"/>
        <Label Content="Nazwa:" Grid.Row="1" Grid.Column="0" Margin="5"/>
        <TextBox Grid.Row="1" Grid.Column="1" Margin="5" Text="{Binding Nazwa}"/>
        <Label Content="Liczba sztuk:" Grid.Row="2" Grid.Column="0" Margin="5"/>
        <TextBox Grid.Row="2" Grid.Column="1" Margin="5"
                 Text="{Binding LiczbaSztuk}"/>
        <Label Content="Magazyn:" Grid.Row="3" Grid.Column="0" Margin="5"/>
        <TextBox Grid.Row="3" Grid.Column="1" Margin="5" Text="{Binding Magazyn}"/>
        <Button x:Name="btnPotwierdz" Grid.Row="5" Grid.Column="0"
                Grid.ColumnSpan="2" Margin="4" MinWidth="120"
                HorizontalAlignment="Center" Content="Potwierdź"
                Click="btnPotwierdz_Click"/>
    </Grid>
```

Panel składa się z 5 wierszy i 2 kolumn, przy czym w ostatnim wierszu, w którym definiowany jest przycisk *Potwierdź*, dwie kolumny zostały połączone (poprzez ustawienie właściwości `Grid.ColumnSpan="2"`). Dla pól tekstowych atrybut `Text` ma przypisaną wartość, umożliwiającą wiązanie z poszczególnymi właściwościami klasy (klasę `Produkt` wykonamy w kolejnym podrozdziale). I tak przykładowo pole tekstowe, w którym będzie wpisywany symbol produktu, ma dla tego atrybutu wartość: `Text="{Binding Symbol}"`, co oznacza, że zostanie powiązane z właściwością `Symbol`. Wykorzystano tu zapis pozwalający pominąć jawne przypisanie właściwości `Path`. Można by użyć także alternatywnego zapisu: `Text="{Binding Path= Symbol}"`. Należy zwrócić uwagę na to, że panel `Grid` ma tu swoją nazwę — `x:Name=" gridProdukt"`. Ta nazwa zostanie wykorzystana w *code-behind* do przypisania kontekstu dla wiązania danych. Wewnątrz panelu tylko jeden element ma nadaną nazwę — przycisk *Potwierdź* (`btnPotwierdz`). Dzięki wiązaniu danych nie ma potrzeby używać nazw pozostałych elementów w *code-behind*.

Dla przedstawionego kodu XAML otrzymamy okno w widoku *Design*, jak na rysunku 5.2.




**Rysunek 5.2.** *Panel Grid aplikacji Produkt*

# 5.3 Definicja klasy Produkt i code-behind

W dalszym kroku stworzymy nową klasę w osobnym pliku[1]. Plik nazwijmy *Produkt.cs*. Należy tam umieścić kod klasy:

```
class Produkt
{
    public string Symbol { get; set; }
    public string Nazwa { get; set; }
    public int LiczbaSztuk { get; set; }
    public string Magazyn { get; set; }

    public Produkt(string sym, string naz, int lszt, string mag)
    {
        Symbol = sym;
        Nazwa = naz;
        LiczbaSztuk = lszt;
        Magazyn = mag;
    }

    public override string ToString()
    {
        return String.Format("{0} {1} {2} {3}", Symbol, Nazwa, LiczbaSztuk,
            Magazyn);
    }
}
```

Kod klasy jest prosty, zawiera cztery publiczne właściwości opisujące produkt, konstruktor oraz metodę `ToString`, zwracającą informacje o produkcie.

W pliku *MainWindow.xaml.cs* należy podmienić kod klasy `MainWindow` na następujący:

```
public partial class MainWindow : Window
{
    private Produkt p1 = null;

    public MainWindow()
    {
        InitializeComponent();
        PrzygotujWiazanie();
    }

    private void PrzygotujWiazanie()
    {
        p1 = new Produkt("DZ-10", "długopis żelowy", 132, "Katowice 1");
        gridProdukt.DataContext = p1;
    }
}
```

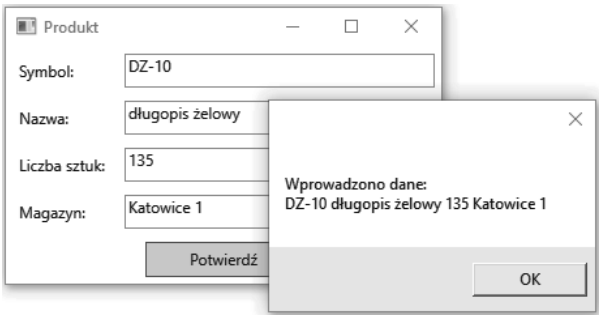
[1] Aby dodać nowy plik w projekcie, należy kliknąć nazwę projektu w oknie *Solution Explorer*, a następnie po naciśnięciu prawego klawisza myszy wybrać opcję *Add/Class*.

W metodzie `PrzygotujWiazanie` tworzony jest obiekt klasy `Produkt` i ustawiany jest kontekst dla wiązania danych. Wiązanie będzie realizowane z kontrolkami tego panelu, którym przypisano (w kodzie XAML) rozszerzenie znaczników `Binding`, np. `Text="{Binding Symbol}"`.

Została nam już tylko ostatnia rzecz do wykonania w tym programie, mianowicie definicja metody obsługującej zdarzenie kliknięcia przycisku *Potwierdź*. Kliknij w tym celu dwa razy lewym klawiszem myszy ten przycisk w widoku *Design* i wpisz kod metody:

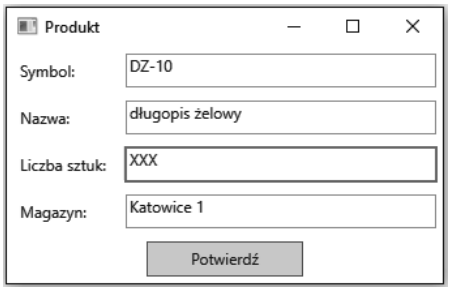
```
private void btnPotwierdz_Click(object sender, RoutedEventArgs e)
{
    string tekst = String.Format("{0}{1}{2}", "Wprowadzono dane:",
        Environment.NewLine, p1.ToString());
    MessageBox.Show(tekst);
}
```

Ponieważ nigdzie nie zapisujemy danych, metodę tę wykorzystamy jedynie do tego, aby wyświetlić informacje o produkcie. Program można już uruchomić. Wstępnie pojawiają się dane dla produktu, można je dowolnie zmienić, a następnie kliknąć przycisk *Potwierdź* (rysunek 5.3).




**Rysunek 5.3.** *Testowanie aplikacji Produkt*

W programie nie wykonaliśmy walidacji, czyli sprawdzania poprawności danych, jakie wprowadza użytkownik. Mimo to pewne możliwości program w tym zakresie posiada. Popatrzmy na rysunek 5.4. Po wpisaniu tekstu **XXX** w polu *Liczba sztuk* i kliknięciu dowolnego innego pola (lub przycisku) obramowanie tego pola zmienia kolor na czerwony.




**Rysunek 5.4.** *Walidacja pola tekstowego*

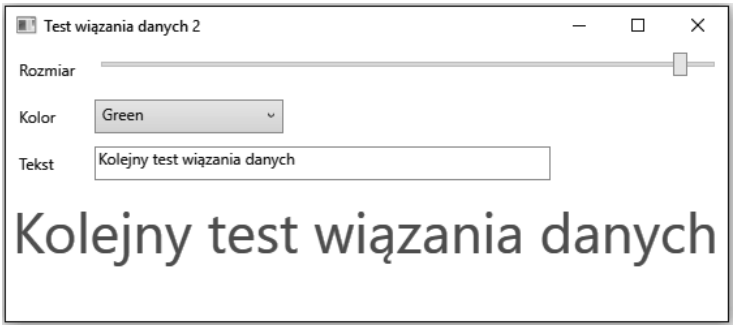
Nie jest możliwa konwersja tekstu `XXX` do danej typu `int`, czyli typu właściwości `LiczbaSztuk` (z klasy `Produkt`) połączonej z tą kontrolką. Wyświetlenie czerwonej ramki pola tekstowego zapewnia domyślny szablon[2], nie zapewnia on jednak stosownego komunikatu. Ponadto nie mamy obecnie możliwości sprawdzania poprawności formatu danych. Przykładowo wiedząc, że symbol produktu składa się z sekwencji dużych liter, łącznika oraz kilku cyfr, moglibyśmy sprawdzić, czy użytkownik wpisał poprawny symbol. Istnieje kilka sposobów przeprowadzenia takiej walidacji danych, wybrane z nich zostaną opisane w rozdziale 12.

# 5.4 Zadania

W pierwszym zadaniu należy dowiązać kilka kontrolek do jednej. Drugi program będzie wymagał zmiany sposobu aktualizacji kontrolki. Natomiast ostatnie zadanie wymaga użycia właściwości służącej do formatowania danej użytej w wiązaniu.

## Zadanie 5.1

Program testujący wiązanie danych z rozdziału 5.1 rozwiń o dwie nowe kontrolki, które mają podlegać wiązaniu: listę rozwijaną z listą kilku kolorów oraz pole tekstowe do wpisywania tekstu. Program ma wyświetlać zadany tekst w wybranym kolorze dla określonego rozmiaru (rysunek 5.5).




**Rysunek 5.5.** *Przykładowe działanie programu*

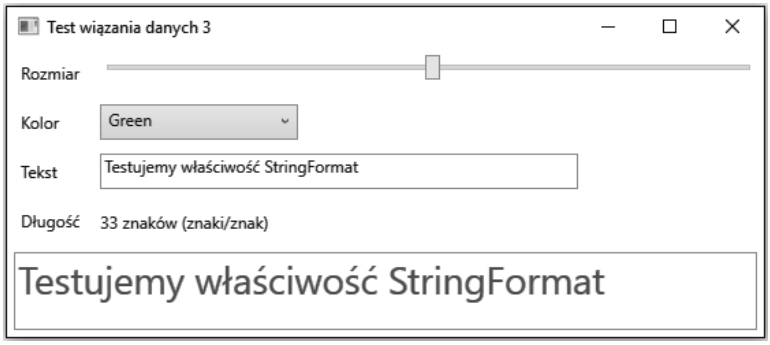
## Zadanie 5.2

Wykonaj nową wersję programu z poprzedniego zadania, zmieniając element docelowy z `TextBlock` na `TextBox`. Po zmianie i uruchomieniu program będzie wyglądał bardzo podobnie, ale będzie widoczna zmiana w działaniu, a mianowicie będzie można zmienić element docelowy wiązania. Efekt tej zmiany jest jednak widoczny dopiero po zmianie fokusu (aktywacji innej kontrolki). Wykonaj taką wersję programu, aby zmiana w kierunku od celu do źródła wiązania była natychmiastowa (podobnie jak to ma miejsce w drugą stronę).

[2] Szablony zostaną przedstawione w rozdziale 11.

### Zadanie 5.3

Uzupełnij program z poprzedniego zadania o nowy element — `TextBlock`, w którym będzie się wyświetlać aktualna liczba znaków tekstu umieszczonego w polu *Tekst*. W polu tym zaraz za liczbą znaków ma się wyświetlić tekst „znaków (znaki/znak)". Wykorzystaj w tym celu właściwość `StringFormat` klasy `Binding`. Przykładowy efekt działania programu przedstawia rysunek 5.6.




**Rysunek 5.6.** *Przykładowe działanie programu*

# 5.5 Wskazówki do zadań

Wskazówki zawierają dość szczegółowe wyjaśniania i przykładowe propozycje rozwiązania.

### Wskazówki do zadania 5.1

W programie mają być trzy kontrolki źródłowe: suwak dla rozmiaru czcionki, lista rozwijana dla koloru czcionki i pole tekstowe do wpisywania tekstu, a także jedna kontrolka dla celu wiązania — blok tekstu. Należałoby jeszcze dodać trzy kontrolki z etykietą opisującą trzy źródłowe komponenty. W szczegółach zmiany względem bazowego programu wyglądałyby następująco: należy powiększyć nieco rozmiar okna, dodać nowe kontrolki, które zalecam ułożyć w panelu `Grid` mającym 4 wiersze i 2 kolumny, a ostatni element (`TextBlock`) powinien zajmować obie kolumny w ostatnim wierszu. Przykład użycia listy rozwijanej (`ComboBox`) został opisany w podrozdziale 3.2. Wpisz do listy angielskie nazwy kolorów (np. `Black`, `Red`, `Green`)[3]. Przykładowa realizacja wiązania danych w definicji znacznika `TextBlock` mogłaby wyglądać tak:

```
<TextBlock Grid.Row ="3" Grid.Column ="0" Grid.ColumnSpan="2" Margin="5"
    FontSize="{Binding Path=Value, ElementName=rozmiarTekstu}"
    Text="{Binding Path=Text, ElementName=txtTekst}"
    Foreground="{Binding Path=SelectedItem.Content, ElementName=cmbKolor}"/>
```

[3] W jednym z dalszych rozdziałów dowiesz się, jak wykonać wiązanie danych z użyciem konwerterów wartości (wówczas nazwy kolorów będą mogły być polskie).

## Wskazówki do zadania 5.2

Możemy wykorzystać kod XAML znacznika `TextBlock` podany we wskazówkach do poprzedniego zadania. W pierwszej kolejności należy podmienić znacznik `TextBlock` na `TextBox`, zostawiając pozostałe ustawienia. To już wystarczy do testowania działania w domyślnym trybie aktualizacji. Zmiana elementu docelowego jest aktualizowana w elemencie źródłowym dopiero po **utracie fokusu** w kontrolce docelowej (czyli sprawieniu, że kontrolka ta przestaje być aktywna). Po przeprowadzeniu testu można przejść do wykonania drugiej części zadania, mianowicie zmiany domyślnego sposobu aktualizacji elementu źródłowego.

W wykonanych dotąd w tym rozdziale programach aktualizacja danych następowała od obiektu źródłowego do obiektu docelowego, ale może przebiegać także w drugim kierunku. W klasie `Binding` jest właściwość `Mode`, która przyjmuje jedną z wartości typu wyliczeniowego `BindingMode`. Typ ten zawiera pięć elementów[4], wśród których są `OneWay` i `TwoWay`. Tryb `OneWay` oznacza, że element docelowy jest aktualizowany przy każdej zmianie źródła, natomiast tryb `TwoWay` oznacza aktualizację obustronną (od źródła do celu i od celu do źródła). Większość kontrolek ma domyślnie przypisany tryb `OneWay`, ale takie kontrolki jak `TextBox`, umożliwiające edycję danych, mają ustawiony domyślnie tryb `TwoWay`.

W przypadku gdy aktualizacja następuje w kierunku od celu do źródła, można ją wykonać na trzy sposoby: natychmiast, po utracie fokusu lub po jawnym wywołaniu metody aktualizującej źródło. Decyduje o tym właściwość `UpdateSourceTrigger`, której można przypisać wartość typu wyliczeniowego `UpdateSourceTrigger`. Wśród elementów tego typu są wartości[5]: `PropertyChanged` (natychmiastowa aktualizacja celu na podstawie źródła) i `LostFocus` (źródło jest aktualizowane po utracie fokusu). Domyślnym ustawieniem sposobu aktualizacji celu dla kontrolki `TextBox` jest `LostFocus`. Na podstawie przytoczonych informacji widzimy, że zmiana w naszym programie będzie niewielka. Wystarczy dodać do wyrażenia `Binding` przypisanie odpowiedniej wartości dla właściwości `UpdateSourceTrigger`. Definicja wiązania dla właściwości `Text` będzie wyglądać następująco:

```
Text="{Binding Path=Text, ElementName=txtTekst,
     UpdateSourceTrigger= PropertyChanged}"
```

## Wskazówki do zadania 5.3

Dodaj w panelu `Grid` nowy wiersz, a następnie dotychczasową lokalizację dla ostatniej kontrolki (`TextBox`) przenieś do nowego ostatniego wiersza (`Grid.Row ="4"`). W wierszu powyższym dodaj w lewej kolumnie etykietę z opisem *Długość*, a w prawej kolumnie zdefiniuj element `TextBlock` według wzoru:

---

[4] Wartości typu wyliczeniowego `BindingMode` („BindingMode Enumeration”): *https://msdn.microsoft.com/en-us/library/system.windows.data.bindingmode(v=vs.110).aspx*.

[5] Wartości typu wyliczeniowego `UpdateSourceTrigger` („UpdateSourceTrigger Enumeration”): *https://msdn.microsoft.com/en-us/library/system.windows.data.updatesourcetrigger(v=vs.110).aspx*.

```
<TextBlock Grid.Row ="3" Grid.Column ="1" HorizontalAlignment="Left"
           Margin="5,11,0,5"
           Text="{Binding StringFormat={}{0} znaków (znaki/znak),
           Path=Text.Length, ElementName=txtTekst}"/>
```

Jeżeli wartość, do której określono wiązanie, ma być wyświetlona w innej postaci niż domyślna, można użyć właściwości `StringFormat` z klasy `Binding`. Jak wspomniałam w podrozdziale 1.4 na temat podstaw XAML, ujęcie wartości atrybutu w nawiasy klamrowe `{}` oznacza, że stanowi ona rozszerzenie znaczników. Jeśli na początku tekstu określającego wartość właściwości chcemy użyć nawiasu klamrowego jako zwykłego literału znakowego, nie traktowanego jako rozszerzenie znaczników, musimy przed nim (po znaku przypisania) użyć pustej pary nawiasów klamrowych. I tak jest właśnie w prezentowanym przykładzie: `StringFormat={}{0} znaków (znaki/znak)`. Pusta para nawiasów klamrowych po znaku przypisania byłaby zbędna, gdyby wartość dla właściwości `StringFormat` zaczynała się od innego znaku niż klamra. Na przykład przetestuj taki wariant: `StringFormat=liczba znków: {0}`. Puste klamry byłyby zbędne także w przypadku, gdybyśmy użyli `Binding` jako elementu XAML. Wówczas kod znacznika dla bloku tekstu mógłby wyglądać tak:

```
<TextBlock Grid.Row ="3" Grid.Column ="1" HorizontalAlignment="Left"
           Margin= "5,11,0,5">
    <TextBlock.Text>
        <Binding Path="Text.Length" ElementName="txtTekst">
            <Binding.StringFormat>
                {0} znaków (znaki/znak)
            </Binding.StringFormat>
        </Binding>
    </TextBlock.Text>
</TextBlock>
```

Kod taki byłby jednak dłuższy i z tego powodu zazwyczaj wybierany jest ten pierwszy wariant.

Zwróć uwagę, że choć definiowana tu kontrolka jest powiązana bezpośrednio tylko z jednym polem tekstowym, to efekt zmian w postaci aktualnej liczby znaków widzimy także wówczas, gdy użytkownik zmienia tekst w drugim polu tekstowym, co jest konsekwencją tego, że oba pola tekstowe są ze sobą powiązane.

Właściwość `StringFormat`, którą wykorzystaliśmy w tym zadaniu, stanowi jedną z możliwości WPF w zakresie definiowania formatu wyświetlanej wartości dla wiązanych danych. W dalszej części książki (w rozdziale 11.) omówię inne możliwości, a mianowicie szablony danych i konwertery.

# Skorowidz

## L



## M

## O



## P



## R



## S



## T



## U



## V



## W

## X



## Z

## Rozpocznij przygodę z programowaniem dla systemu Windows!

Technologia Windows Presentation Foundation firmy Microsoft to jedno z najlepszych rozwiązań do tworzenia aplikacji działających pod kontrolą systemu Windows. WPF integruje interfejs użytkownika, grafikę, multimedia i dokumenty oraz ułatwia implementację wzorców projektowych, które oddzielają warstwę logiczną od warstwy prezentacji. Dużą zaletę tej technologii stanowi możliwość kompozycji i adaptacji poszczególnych elementów, z których budowany jest interfejs.

Jeśli uczysz się programowania od niedawna i chcesz tworzyć aplikacje desktopowe przy użyciu technologii WPF, to książka dla Ciebie! Z jej pomocą szybko rozpoczniesz pisanie programów. W ten sposób, tworząc kolejne aplikacje okienkowe, będziesz poznawać kluczowe możliwości tej platformy.

Podręcznik nie przedstawia wszystkich podstawowych funkcji zapewnianych przez WPF, ponieważ nie są one potrzebne na początkowym etapie nauki. Opracowanie prostej aplikacji o charakterze biznesowym nie wymaga wykorzystania grafiki 3D czy animacji, nieodzowna jest tu natomiast znajomość podstawowych kontrolek, paneli oraz kluczowych mechanizmów, takich jak wiązanie danych. I właśnie tego nauczysz się, pisząc z tą książką swoje pierwsze programy z użyciem WPF!

- Podstawy języka XAML
- Wykorzystanie podstawowych kontrolek i standardowych paneli
- Wiązanie danych, wyzwalacze, konwertery, zasoby, style i szablony
- Wybrane sposoby walidacji danych
- Podstawowe informacje na temat wzorca projektowego MVVM

## Sięgnij po doskonały przewodnik dla początkujących programistów!

**ANNA KEMPA** jest autorką kursu *Język C#. Kurs video. Poziom pierwszy. Programowanie dla początkujących* dostępnego na stronie videopoint.pl oraz współautorką podręcznika *Wstęp do programowania w C#. Łatwy podręcznik dla początkujących.*

Helion

księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

ISBN 978-83-283-3272-0
9 788328 332720

Informatyka w najlepszym wydaniu
cena: 39,90 zł