Microsoft

John Sharp

Microsoft Visual C# 2015
Krok po kroku

Wydanie ósme

Przekład: Natalia Chounlamany, Janusz Machowski, Krzysztof Szkudlarek,
Marek Włodarz

APN Promise, Warszawa 2016

Microsoft Visual C# 2015 Krok po kroku

Authorized Polish translation of the English language edition entitled: Microsoft Visual C# Step by Step, 8th Edition, ISBN 978-1-5093-0104-1, by John Sharp, published by Pearson Education, Inc, publishing as Microsoft Press, A Division Of Microsoft Corporation.

Polish language edition published by APN PROMISE SA Copyright © 2016

Autoryzowany przekład z wydania w języku angielskim, zatytułowanego: Microsoft Visual C# Step by Step, 8th Edition, ISBN 978-1-5093-0104-1, by John Sharp, opublikowanego przez Pearson Education, Inc, publikującego jako Microsoft Press, oddział Microsoft Corporation.

APN PROMISE SA, biuro: ul. Domaniewska 44a, 02-672 Warszawa, tel. +48 22 35 51 600, fax +48 22 35 51 699 e-mail: mspress@promise.pl

ISBN: 978-83-7541-179-9

Przekład: Natalia Chounlamany, Janusz Machowski, Krzysztof Szkudlarek, Marek Włodarz

Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Spis treści

Wstęp

Język Microsoft Visual C# jest bardzo potężnym, a jednocześnie prostym językiem programowania, przeznaczonym głównie dla programistów, którzy tworzą aplikacje oparte na platformie Microsoft .NET Framework. Visual C# odziedziczył wiele z najlepszych cech języka C++ oraz Microsoft Visual Basic i tylko kilka występujących w tych językach niespójności lub anachronizmów, co zaowocowało powstaniem bardziej przejrzystego i bardziej logicznego języka programowania. Język C# w wersji 1.0 miał swój publiczny debiut w roku 2001. Pojawienie się wersji C# 2.0 wraz z programem Visual Studio 2005 oznaczało dodanie do tego języka kilku ważnych i nowych funkcji, takich jak ogólne typy wyliczeniowe i metody anonimowe. W wersji C# 3.0, która pojawiła się wraz z opublikowaniem programu Visual Studio 2008, dodana została obsługa metod rozszerzających, wyrażeńlambda oraz najważniejszej ze wszystkich nowości – obsługi zapytań w języku LINQ (Language Integrated Query). Wprowadzona w roku 2010 wersja C# 4.0 zaoferowała kolejne udoskonalenia w zakresie polepszenia możliwości współdziałania z innymi technologiami i językami programowania. Funkcje te obejmowały obsługę argumentów nazwanych i opcjonalnych, typ *dynamic*, którego użycie wskazywało, że środowisko uruchomieniowe powinno zastosować dla danego obiektu tzw. późne wiązanie (ang. late binding). Bardzo ważną zmianą wprowadzoną do wersji platformy .NET Framework, opublikowanej w tym samym czasie co wersja C# 4.0, były klasy i typy danych składające się na nową bibliotekę równoległego realizowania zadań – TPL (Task Parallel Library). Korzystając z biblioteki TPL można tworzyć wysoko skalowalne aplikacje, które będą w pełni wykorzystywaćmożliwości wielordzeniowych procesorów. W najnowszej wersji języka C# 5.0 dodano natywną obsługę dla przetwarzania zadańw sposób asynchroniczny poprzez użycie modyfikatora metody *async* oraz operatora *await*. Wersja C# 6.0 wprowadza kolejne ulepszenia zaprojektowane z myślą o ułatwianiu życia programistom. Te udoskonalenia to między innymi interpolacja łańcuchów (już nigdy nie będziemy musieli korzystać z *String.Format*!), ulepszone sposoby implementacji właściwości czy metody wcielające wyrażenia. Funkcje te zostaną opisane w niniejszej książce.

Innym ważnym wydarzeniem dla firmy Microsoft jest premiera systemu Windows 10. Ta nowa wersja systemu Windows łączy w sobie najlepsze (i najbardziej lubiane) aspekty poprzednich wersji systemu operacyjnego z obsługą aplikacji o wysokim stopniu interaktywności, które mogą współdzielić pomiędzy sobą dane i współpracować ze sobą nawzajem lub łączyć się z usługami działającymi w chmurze. Kluczowy element wersji Windows 10 stanowią aplikacje Universal Windows Platform (UWP) – zaprojektowane tak, aby mogły być uruchamiane na dowolnym urządzeniu Windows 10, począwszy od w bogato wyposażonego komputera, po laptop, tablet, smartfon,

a nawet urządzenia IoT (Internet of Things) z ograniczonymi zasobami. Po opanowaniu podstawowych funkcji języka C#, ważne jest zdobycie umiejętności budowania aplikacji, które mogą być uruchamiane na wszystkich wspomnianych platformach.

Aktywacja głosowa to kolejna funkcja, która zyskała na popularności. System Windows 10 oferuje Cortanę, czyli osobistą asystentkę cyfrową, która może być aktywowana za pomocą głosu*. Możemy zintegrować swoje aplikacje z Cortaną, aby zapewnić im możliwość uczestniczenia w wyszukiwaniu danych i innych operacjach. Mimo komplikacji związanych zazwyczaj z analizą mowy, przygotowanie aplikacji do reagowania na żądania Cortany jest zaskakująco proste i zostało omówione w rozdziale 26. Ponadto chmura stała się tak istotnym elementem architektury wielu systemów, począwszy od dużych systemów korporacyjnych po aplikacje mobilne działające na smartfonach użytkowników, że postanowiliśmy poświęcić temu aspektowi rozwoju oprogramowania ostatni rozdział książki.

Środowisko programowania oferowane przez pakiet Visual Studio 2015 sprawia, że korzystanie ze wszystkich tych nowych i potężnych funkcji jest bardzo łatwe, a wiele nowych kreatorów i ulepszeń wprowadzonych do najnowszej wersji Visual Studio pozwala znacząco podnieść produktywność programistów. Mamy nadzieję, że korzystanie z tej książki będzie równie przyjemne, jak jej pisanie!

Dla kogo przeznaczona jest ta książka

Książka ta powstała przy założeniu, że Czytelnik ma już pewne doświadczenie w programowaniu i pragnie poznać podstawy programowania w języku C# przy użyciu programu Visual Studio 2015 oraz platformy .NET Framework w wersji 4.6. Po przeczytaniu tej książki jej Czytelnicy powinni dysponować dobrą znajomością języka C# i powinni umieć używać tego języka do tworzenia szybkich i skalowalnych aplikacji dla systemu operacyjnego Windows 10.

Dla kogo nie jest przeznaczona ta książka

Niniejsza książka skierowana jest do osób, które dopiero uczą się języka C#, ale nie są nowicjuszami w programowaniu jako takim. Dlatego koncentruje się ona głównie na kwestiach związanych z samym językiem C#. Celem tej książki nie jest dostarczenie wyczerpującego omówienia rozlicznych technologii pozwalających na tworzenie aplikacji przeznaczonych do użytku w dużych przedsiębiorstwach, takich jak ADO.NET, ASP.NET, Windows Communication Foundation lub Windows Workflow Foundation. Czytelnicy oczekujący większej ilości informacji na temat jednej z tych technologii powinni rozważyć lekturę kilku innych tytułów wydanych przez Microsoft Press.

---

* przyp. tłum. Funkcja Cortana nie jest jeszcze dostępna w polskiej wersji systemu Windows 10.

Organizacja książki

Niniejsza książka została podzielona na następujące cztery części:

- Część I, zatytułowana „Wprowadzenie do języka Microsoft Visual C# oraz programu Microsoft Visual Studio 2015" stanowi wprowadzenie do podstawowej składni języka C# oraz środowiska programowania Visual Studio.
- Część II, zatytułowana „Omówienie modelu obiektowego języka C#", przedstawia więcej szczegółów związanych z tworzeniem i zarządzaniem w języku C# nowymi typami danych, a także wyjaśnia, w jaki sposób należy zarządzać zasobami wskazywanymi przez te typy danych.
- Część III, zatytułowana „Tworzenie rozszerzalnych typów danych w języku C#", zawiera poszerzone omówienie tych elementów oferowanych przez język C#, które można wykorzystywać do tworzenia typów danych nadających się do używania w wielu różnych aplikacjach.
- Część IV, zatytułowana „Tworzenie aplikacji Universal Windows Platform", opisuje uniwersalny model programowania w systemie Windows 10 i wyjaśnia, jak używać języka C# do tworzenia interaktywnych aplikacji opartych na tym nowym modelu.

Określenie najlepszego miejsca, od którego należy rozpocząć lekturę tej książki

Niniejsza książka ma za zadanie ułatwić jej Czytelnikom podniesienie swoich umiejętności w kilku podstawowych obszarach. Z książki tej mogą korzystać zarówno początkujący programiści, jak również programiści mający już pewne doświadczenie w innych językach programowania, takich jak C, C++, Java lub Visual Basic. Zamieszczona dalej tabela powinna ułatwić każdemu określenie najlepszego miejsca, od którego należy rozpocząć lekturę tej książki.

Jeżeli jesteś	Wykonaj następujące kroki
Początkującym programistą w dziedzinie programowania zorientowanego obiektowo	1. Zainstaluj pliki używane w opisywanych w tej książce ćwiczeniach, zgodnie z opisem podanym w dalszej części, zatytułowanej „Przykładowe kody źródłowe". 2. Przeczytaj po kolei wszystkie rozdziały z części I, II i III. 3. Przeczytaj odpowiednie rozdziały z części IV, stosownie do poziomu swojego doświadczenia oraz poziomu swoich potrzeb.

Jeżeli jesteś	Wykonaj następujące kroki
Programistą dobrze obeznanym z proceduralnymi językami programowania, takimi jak np. język C, ale nie znasz jeszcze języka C#	1. Zainstaluj pliki używane w opisywanych w tej książce ćwiczeniach, zgodnie z opisem podanym w dalszej części, zatytułowanej „Przykładowe kody źródłowe”. .2. Zapoznaj się pobieżnie z treścią pierwszych pięciu rozdziałów, aby uzyskać ogólny obraz języka C# oraz możliwości programu Visual Studio 2015, a następnie skoncentruj się na rozdziałach od 6 do 22. 3. Przeczytaj odpowiednie rozdziały z części IV, stosownie do poziomu swojego doświadczenia oraz poziomu swoich potrzeb.
Programistą mającym już doświadczenie w innych zorientowanych obiektowo językach programowania, takich jak np. C++ lub Java, i pragnącym nauczyć się języka C#	1. Zainstaluj pliki używane w opisywanych w tej książce ćwiczeniach, zgodnie z opisem podanym w dalszej części, zatytułowanej „Przykładowe kody źródłowe”. 2. Zapoznaj się pobieżnie z treścią pierwszych siedmiu rozdziałów, aby uzyskać ogólny obraz języka C# oraz możliwości programu Visual Studio 2015, a następnie skoncentruj się na rozdziałach od 7 do 22. 3. Przeczytaj rozdziały z części IV, aby dowiedzieć się, w jaki sposób tworzyć aplikacje Universal Windows Platform.
Programistą znającym język Visual Basic 6 i pragnącym nauczyć się języka C#	1. Zainstaluj pliki używane w opisywanych w tej książce ćwiczeniach, zgodnie z opisem podanym w dalszej części, zatytułowanej „Przykładowe kody źródłowe”. 2. Przeczytaj po kolei wszystkie rozdziały z części I, II i III. 3. Przeczytaj rozdziały z części IV, aby dowiedzieć się, w jaki sposób tworzyć aplikacje Universal Windows Platform. 4. Przeczytaj krótkie powtórzenia, znajdujące się na końcu każdego rozdziału, aby szybko uzyskać potrzebne informacje na temat konkretnych konstrukcji języka C# oraz funkcji programu Visual Studio 2015.
Zainteresowany znalezieniem potrzebnych informacji, związanych z prezentowanymi w tej książce ćwiczeniami	1. Skorzystaj z indeksu lub spisu treści, aby znaleźć potrzebne informacje na konkretny temat. 2. Przeczytaj krótkie powtórzenia, znajdujące się na końcu każdego rozdziału, aby szybko zapoznać się z omawianymi w danym rozdziale technikami oraz elementami składni.

Większość rozdziałów tej książki zawiera użyteczne przykłady ułatwiające zrozumienie omawianych koncepcji i pojęć. Każdy Czytelnik, niezależnie od tego, które części tej książki będą dla niego najbardziej interesujące, powinien koniecznie pobrać i zainstalować na swoim komputerze omawiane aplikacje przykładowe.

Konwencje i cechy charakterystyczne stosowane w tej książce

Zawarte w tej książce informacje są prezentowane przy użyciu konwencji poprawiających czytelność oraz ułatwiających śledzenie toku narracji.

- Każde ćwiczenie składa się z serii zadań, prezentowanych jako seria ponumerowanych kroków (1, 2, itd.) zawierających opis wszystkich działań niezbędnych do ukończenia danego ćwiczenia.
- Elementy umieszczone w ramkach z ikoną na marginesie i etykietą, taką jak np. „Uwaga", zawierają dodatkowe informacje lub opis alternatywnego sposobu wykonania danego kroku.
- Tekst, który powinien zostać wpisany przez Czytelnika (poza blokami kodu), wyróżniany jest za pomocą wytłuszczonej czcionki.
- Znak plus (+) występujący pomiędzy nazwami dwóch klawiszy oznacza, że klawisze te należy wcisnąć równocześnie. Np. zdanie „Wciśnij klawisze Alt+Tab" oznacza, że najpierw należy wcisnąć klawisz Alt, a następnie trzymając ten klawisz wciśnięty wcisnąć klawisz Tab.

Wymagania systemowe

Do wykonania prezentowanych w tej książce ćwiczeń potrzebny będzie komputer spełniający następujące wymagania sprzętowe i programowe:

- System operacyjny Windows 10 edycja Professional (lub wyższa).
- Edycja Visual Studio Community 2015, Visual Studio Professional 2015 lub Visual Studio Enterprise 2015.

Ważne Trzeba zainstalować narzędzie programistyczne Windows 10 w Visual Studio 2015.

- Komputer z procesorem 1.6 GHz lub szybszym (zaleca się 2 GHz).
- 1 GB (w systemach 32-bitowych) lub 2 GB (w systemach 64-bitowych) pamięci RAM (w przypadku pracy na maszynie wirtualnej, wielkości te należy powiększyć o dodatkowe 512 MB).

- 10 GB wolnego miejsca na dysku twardym.
- Dysk twardy o prędkości obrotowej 5400 RPM lub szybszy.
- Karta graficzna kompatybilna ze standardem DirectX 9, o rozdzielczości 1024 × 768 lub wyższej.
- Napęd optyczny DVD-ROM (w przypadku instalowania oprogramowania Visual Studio z płyty DVD).
- Połączenie z publiczną siecią Internet, wymagane do pobrania oprogramowania lub przykładów dla poszczególnych rozdziałów.

W zależności od konfiguracji używanego systemu Windows, zainstalowanie i skonfigurowanie oprogramowania Visual Studio 2015 może wymagać posiadania uprawnień lokalnego administratora.

Ponadto trzeba włączyć na komputerze tryb dewelopera, aby móc tworzyć i uruchamiać aplikacje UWP. Dodatkowe informacje dotyczące osiągania tego celu znaleźć można w artykule „Enable Your Device for Development" (w jęz. angielskim) o adresie *https://msdn.microsoft.com/library/windows/apps/dn706236.aspx*.

Przykładowe kody źródłowe

W większości rozdziałów tej książki zamieszczone zostały ćwiczenia pozwalające na interaktywne wypróbowanie nowych umiejętności, nabytych podczas lektury danego rozdziału. Wszystkie te przykładowe projekty można pobrać w wersji wyjściowej (tj. takiej, od której rozpoczyna się dane ćwiczenie) lub w wersji końcowej (tj. takiej, jaką otrzymalibyśmy po starannym wykonaniu całego ćwiczenia) z następującej strony webowej:

http://aka.ms/sharp8e/companioncontent

Uwaga Oprócz pobrania przykładowych kodów źródłowych, należy pamiętać także o konieczności zainstalowania w systemie pakietu oprogramowania Visual Studio 2015. Należy także zainstalować najnowsze wersje pakietów serwisowych, które będą dostępne dla oprogramowania Visual Studio oraz dla systemu operacyjnego Windows.

Instalowanie przykładowych kodów źródłowych

Aby zainstalować na komputerze przykładowe kody źródłowe, które umożliwią praktyczne wykonywanie opisywanych w tej książce ćwiczeń, należy wykonać następujące kroki.

1. Rozpakuj plik CSharpSBS.zip pobrany ze strony web powiązanej z książką do swojego katalogu Dokumenty.
2. Zaakceptuj postanowienia licencyjne w wyświetlonym monicie.

Uwaga Jeśli podczas wykonywania tych operacji nie zostanie wyświetlony tekst umowy licencyjnej, to z treścią tej umowy można zapoznać się na tej samej stronie webowej, z której pobrany został plik z przykładowymi kodami źródłowymi.

Korzystanie z przykładowych kodów źródłowych

Wszystkie rozdziały tej książki zawierają dokładne instrukcje wyjaśniające, kiedy i w jaki sposób należy korzystać z przykładowych kodów źródłowych dla danego rozdziału. Gdy nadejdzie pora użycia kolejnego przykładu, podane zostaną dokładne instrukcje, w jaki sposób należy otworzyć odpowiednie pliki.

Ważne Niektóre projekty są zależne od pakietów NuGet, które nie zostały dołączone do przykładów kodu. Te pakiety zostają automatycznie pobrane w trakcie pierwszej kompilacji projektu. W konsekwencji, jeśli Czytelnik otworzy projekt i zacznie go analizować przed kompilacją, Visual Studio może zgłaszać wiele błędów spowodowanych nierozwiązanymi odwołaniami. Po skompilowaniu projektu problemy powinny zostać rozwiązane i błędy powinny zniknąć

Dla tych Czytelników, którzy chcieliby poznać więcej szczegółów, poniżej zamieszczona została lista wszystkich przykładowych projektów i rozwiązań programu Visual Studio 2015, pogrupowanych według katalogów, w których się one znajdują. W wielu przypadkach przykładowe projekty dostępne są w wersji z plikami w stanie początkowym, umożliwiającym samodzielne przeprowadzenie danego ćwiczenia zgodnie z podanym opisem oraz w wersji końcowej, której można używać jako punktu odniesienia. Gotowe wersje projektów z każdego rozdziału znajdują się w folderach o nazwie uzupełnionej przyrostkiem „-Complete” (Gotowe).

Projekt	Opis
Chapter 1	
TextHello	Pierwszy projekt w języku C#. Projekt ten demonstruje kolejne kroki procesu tworzenia prostego programu wyświetlającego tekst pozdrowienia.
Hello	Ten projekt otwiera okno, które prosi użytkownika o podanie imienia, a następnie wyświetla spersonalizowane powitanie.
Chapter 2	
PrimitiveDataTypes	Projekt demonstrujący sposób deklarowania zmiennych każdego z podstawowych typów danych, sposób przypisywania tym zmiennym wartości oraz sposób wyświetlania wartości tych zmiennych w oknie programu.
MathsOperators	Projekt wprowadzający operatory arytmetyczne (+ – * / %).
Chapter 3	
Methods	Projekt polegający na ponownym przeanalizowaniu kodu poprzedniego projektu MathsOperators i wykorzystaniu metod do uporządkowania kodu źródłowego.
DailyRate	Projekt demonstrujący proces pisania własnych metod, ich uruchamiania oraz krokowego śledzenia przy pomocy debugera z programu Visual Studio 2015.
DailyRate Using Optional Parameters	Projekt demonstrujący sposób definiowania metod akceptujących parametry opcjonalne oraz wywoływania tych metod przy użyciu nazwanych argumentów.
Chapter 4	
Selection	Projekt demonstrujący kaskadowe użycie instrukcji if do zaimplementowania złożonej logiki, polegającej np. na porównywaniu dwóch dat.
SwitchStatement	Prosty program wykorzystujący instrukcję switch do przekształcania znaków na ich reprezentację w formacie XML.
Chapter 5	
WhileStatement	Projekt demonstrujący użycie instrukcji while do odczytywania kolejnych linii z pliku źródłowego i wyświetlania każdej z nich w umieszczonym na formularzu, osobnym polu tekstowym.

Projekt	Opis
DoStatement	Projekt wykorzystujący instrukcję do do przekształcenia liczby dziesiętnej na jej reprezentację ósemkową.
Chapter 6	
MathsOperators	Projekt pokazujący na przykładzie projektu MathsOperators z rozdziału 2, zatytułowanego „Zmienne, operatory i wyrażenia", jak różne nieobsłużone wyjątki mogą doprowadzić do przerwania działania programu. Stabilność działania aplikacji zostaje poprawiona poprzez użycie słów kluczowych try i catch.
Chapter 7	
Classes	Projekt demonstrujący podstawy definiowania własnych klas, uzupełniania ich o publiczne konstruktory, metody oraz pola prywatne. Projekt ten demonstruje również sposób tworzenia nowych instancji klasy za pomocą słowa kluczowego new oraz sposób definiowania statycznych pól i metod.
Chapter 8	
Parameters	Program wyjaśniający różnicę pomiędzy parametrami typu wartościowego a parametrami typu referencyjnego. Program ten demonstruje także użycie słów kluczowych ref i out.
Chapter 9	
StructsAndEnums	Projekt definiujący strukturalny typ danych (przy użyciu słowa kluczowego struct), służący do reprezentowania daty kalendarzowej.
Chapter 10	
Cards	Projekt demonstrujący zastosowanie tablic do zamodelowania puli kart w grze karcianej.
Chapter 11	
ParamsArray	Projekt demonstrujący użycie słowa kluczowego params do tworzenia pojedynczej metody mogącej akceptować dowolną liczbę argumentów typu int.
Chapter 12	
Vehicles	Projekt wykorzystujący interfejsy do utworzenia prostej hierarchii klas pojazdów. Projekt ten demonstruje także sposób definiowania metod wirtualnych.
ExtensionMethod	Projekt demonstrujący sposób tworzenia metody rozszerzającej dla typu int, której zadaniem będzie przekształcanie dziesiętnej liczby całkowitej w jej reprezentację w innym systemie liczenia.

Projekt	Opis
Chapter 13	
Drawing	Projekt implementujący część pakietu graficznego. Projekt ten wykorzystuje interfejsy do zdefiniowania metod udostępnianych i implementowanych przez różne figury geometryczne.
Drawing Using Interfaces	Projekt stanowi wstęp do rozszerzania projektu Drawing poprzez utworzenie klas abstrakcyjnych, oferujących funkcjonalność wspólną dla różnych figur geometrycznych.
Chapter 14	
GarbageCollectionDemo	Projekt demonstrujący sposób bezpiecznego zwalniania zasobów w środowisku wielowątkowym poprzez odpowiednie używanie wzorca Dispose.
Chapter 15	
Drawing Using Properties	Projekt rozszerzający aplikację w projekcie Drawing opracowanym w rozdziale 13 poprzez hermetyzację wybranych danych wewnątrz klasy przy użyciu właściwości.
AutomaticProperties	Projekt demonstrujący sposób tworzenia automatycznych właściwości klas oraz wykorzystywania ich do inicjalizowania nowych instancji klasy.
Chapter 16	
Indexers	Projekt demonstrujący zastosowanie dwóch obiektów indeksujących (indeksatorów): jednego służącego do wyszukiwania numeru telefonu osoby o podanym nazwisku i drugiego służącego do wyszukiwania nazwiska osoby o podanym numerze telefonu.
Chapter 17	
BinaryTree	Rozwiązanie demonstrujące sposób używania ogólnych typów danych do utworzenia struktury bezpiecznej pod względem typu, mogącej zawierać elementy dowolnego typu.
BuildTree	Projekt demonstrujący sposób użycia ogólnych typów danych do zaimplementowania metody bezpiecznej pod względem typu, mogącej akceptować parametry dowolnego typu.
Chapter 18	
Cards	Projekt zawierający zmodyfikowaną wersję kodu z rozdziału 10, demonstrujący sposób użycia kolekcji do zamodelowania puli kart rozdanych pomiędzy graczy w grze karcianej.

Projekt	Opis
Chapter 19	
BinaryTree	Projekt demonstrujący sposób zaimplementowania ogólnego interfejsu typu IEnumerator<T>, który umożliwia utworzenie obiektu wyliczeniowego dla ogólnej klasy Tree.
IteratorBinaryTree	Rozwiązanie wykorzystujące iterator do wygenerowania typu wyliczeniowego dla ogólnej klasy Tree.
Chapter 20	
Delegates	Projekt demonstrujący sposób wykorzystania delegacji do oddzielenia metody od logiki korzystającej z tej metody aplikacji. Następnie projekt zostaje rozszerzony, aby zademonstrować sposób wykorzystania zdarzeń do powiadamiania obiektów o ważnych wydarzeniach oraz sposób przechwytywania tego typu zdarzeń i wykonywania związanych z nimi akcji.
Chapter 21	
QueryBinaryTree	Projekt demonstrujący sposób pobierania danych z obiektu typu drzewo binarne, przy użyciu zapytań w języku LINQ.
Chapter 22	
ComplexNumbers	Projekt definiujący nowy typ danych, modelujący liczby złożone i implementujący klika typowych operatorów używanych dla tego typu danych.
Chapter 23	
GraphDemo	Projekt generujący skomplikowany wykres i wyświetlający go na formularzu UWP. W tym projekcie niezbędne obliczenia wykonywane są przez jeden wątek.
Parallel GraphDemo	Jest to wersja projektu GraphDemo, w którym do wyodrębnienia procesu tworzenia i zarządzania zadaniami użyta została klasa Parallel.
GraphDemo With Cancellation	Projekt pokazujący, jak zaimplementować możliwość zatrzymywania zadań w kontrolowany sposób, zanim same zakończą swoje działanie.
ParallelLoop	Przykładowa aplikacja pokazująca, kiedy nie należy używać klasy Parallel do tworzenia i uruchamiania kilku zadań równocześnie.

Projekt	Opis
Chapter 24	
GraphDemo	Jest to wersja projektu GraphDemo z rozdziału 23, w której w celu asynchronicznego wykonania obliczeń potrzebnych do wygenerowania wykresu zastosowano słowo kluczowe async oraz operator await.
PLINQ	Projekt demonstrujący kilka przykładów wykorzystywania technologii PLINQ do pobierania danych, przy użyciu zadań równoległych.
CalculatePI	Projekt wykorzystujący algorytm próbkowania statystycznego do obliczenia przybliżonej wartości liczby pi. Projekt ten wykorzystuje zadania równoległe.
Chapter 25	
Customers	Ten projekt implementuje skalowalny interfejs użytkownika, który poddaje się skalowaniu dla różnych rozdzielczości ekranu i różnych kształtów formularza. Interfejs użytkownika wykorzystuje style XAML do zmiany czcionki oraz wyświetlanego przez aplikację obrazu tła.
Chapter 26	
DataBinding	Ta wersja projektu Customers wyświetla informacje o klientach, pobierając je ze źródła danych przy użyciu mechanizmu wiązania danych. Projekt ten demonstruje również sposób implementacji interfejsu *INotifyPropertyChanged*, pozwalającego na aktualizowanie informacji o kliencie z poziomu interfejsu użytkownika i odsyłanie wykonanych zmian z powrotem do źródła danych.
ViewModel	W tej wersji projektu Customers zaimplementowano metodologię programowania MVVM (Model-View-ViewModel), co pozwoliło na oddzielenie interfejsu użytkownika od logiki pobierającej dane ze źródła danych.
Cortana	Ten projekt integruje aplikację Customers z funkcją Cortana. Użytkownik będzie mógł przy pomocy poleceń głosowych wyszukiwać klientów według ich nazwiska.

Projekt	Opis
Chapter 27	
Web Service	Rozwiązanie obejmujące aplikację web dostarczającą usługi typu ASP.NET Web Service, używanej przez aplikację Customers do pobierania danych klientów z bazy danych serwera SQL Server. Podczas dostępu do bazy danych usługa web używa modelu encji utworzonego przy użyciu technologii Entity Framework.

Podziękowania

Pomimo faktu, że na okładce tej książki figuruje moje nazwisko, to stworzenie tego rodzaju książki z pewnością nie jest zadaniem dla jednego człowieka. Chciałbym w tym miejscu podziękować wymienionym poniżej osobom za ich bezgraniczne wsparcie i pomoc w realizacji tego zadania.

W pierwszej kolejności swoje podziękowania kieruję do Devona Musgrave w wydawnictwie Microsoft Press, który przebudził mnie z pisarskiego letargu (w rzeczywistości byłem dość zajęty przygotowywaniem materiałów dla Microsoft Patterns & Practices, ale udało mi się wziąć przedłużony urlop na czas pracy nad tą edycją książki). Zaczepiał mnie, przekabacał i uświadamiał rychłe nadejście wersji Windows 10 i Visual Studio 2015, przygotował kontrakt z uzgodnionymi terminami i nie spoczął, dopóki nie podpisałem go własną krwią!

Chciałbym podziękować również Jasonowi Lee, mojemu byłemu podwładnemu, a teraz przełożonemu w firmie Content Master (to dość skomplikowana historia, ale wygląda na to, że znalazł on pewne interesujące negatywy, które nieopatrznie zostawiłem na wierzchu). Jason zajął się mozolnym zadaniem generowania nowych zrzutów ekranu i sprawdzaniem, czy kod około dwudziestu pierwszych rozdziałów został zaktualizowany (i poprawiony). Jeśli w tej edycji pojawią się jakieś błędy, chętnie zrzuciłbym winę na niego, ale oczywiście jako osoba dokonująca końcowej redakcji ponoszę pełną odpowiedzialność.

Podziękowania należą się także Marcowi Young, któremu w udziale przyszło dość żmudne zadanie analizowania mojego kodu w celu upewnienia się, że może on zostać skompilowany i uruchomiony. Jego porady okazały się bardzo pomocne.

Oczywiście, podobnie jak u wielu programistów, pomimo znajomości opisywanych technologii mój język zapewne nie zawsze jest tak płynny i przejrzysty, jak można by oczekiwać. Dlatego chciałem podziękować Johnowi Pierce za korektę błędów gramatycznych i ortograficznych oraz za ogólną poprawę czytelności prezentowanego materiału, dzięki której stał się on bardziej zrozumiały.

Na zakończenie, muszę podziękować mojej biednej żonie Dianie, która obawiała się, że powoli tracę rozum (co mogło być prawdą), gdy zacząłem mamrotać pod nosem różne dziwne frazy do laptopa, aby przekonać Cortanę do współpracy z moją aplikacją. Diana myślała, że non stop dzwonię do „Orlanda Gee", ponieważ dość głośno wykrzykiwałem to imię i nazwisko (przykładowe dane klienta wykorzystywane przeze mnie w ćwiczeniach tej książki). Niestety ze względu na mój akcent, Cortana myślała, że chodzi mi o Orlanda T, Orlanda Key lub nawet Orlanda Quay, co zmusiło mnie w końcu do zmiany imienia i nazwiska stosowanego w ćwiczeniach na Brian Johnson. Zasłyszałem nawet fragment rozmowy Diany z Jasonem, dekoratorem zajmującym się malowaniem naszego holu, w której rozważała ona możliwość przekształcenia jednego z pokoi w izolatkę obitą poduszkami, tak bardzo niepokoił ją mój stan psychiczny! Ale to już „water under the bridge" (upłynęło, jak woda pod mostem) lub „water under the breach" (woda pod wyłomem), według Cortany interpretującej mój specyficzny akcent, który wywodzi się częściowo z Gloucester i częściowo z Kent.

Kończąc zakończenie, muszę wspomnieć moją córkę Francescę – w przeciwnym przypadku poczułaby się ona ogromnie urażona. Chociaż nadal mieszka z nami, jest już całkiem dorosła i pracuje dla firmy programistycznej w Cam, Gloucestershire (nie wspominam ich nazwy, ponieważ jeszcze nie otrzymałem od nich żadnych gratisów).

Errata i wsparcie techniczne

Dołożono wszelkich starań, mających na celu zapewnienie dokładności tej książki oraz towarzyszących jej treści. Informacje o wszelkich błędach, które zostały dostrzeżone i zgłoszone już po opublikowaniu tej książki, dostępne są na stronie wydawnictwa Microsoft Press:

http://aka.ms/sharp8e/errata

W przypadku wykrycia nowego błędu można go zgłosić za pomocą tej samej, wymienionej powyżej strony.

W razie potrzeby uzyskania dodatkowej pomocy technicznej związanej z tą książką prosimy o skontaktowanie się z działem pomocy technicznej wydawnictwa Microsoft Press Book Support poprzez wysłanie wiadomości email na adres *mspinput@microsoft.com*.

Prosimy pamiętać, że pod podanymi adresami nie jest oferowana pomoc techniczna dla oprogramowania firmy Microsoft.

Oczekujemy na uwagi Czytelników

Satysfakcja Czytelników jest najwyższym priorytetem wydawnictwa Microsoft Press i dlatego komentarze Czytelników są dla nas niezwykle cenne. Prosimy o dzielenie się swoimi opiniami na temat tej książki pod adresem:

http://aka.ms/tellpress

Pod podanym powyżej adresem dostępna jest krótka ankieta i zapewniamy, że czytamy wszystkie komentarze i pomysły naszych Czytelników. Z góry dziękujemy za czas poświęcony na jej wypełnienie!

Pozostańmy w kontakcie

Pozostań w kontakcie z nami! Wydawnictwo Microsoft Press obecne jest na Twitterze pod następującym adresem: *http://twitter.com/MicrosoftPress*

CZĘŚĆ I

Wprowadzenie do języka Microsoft Visual C# oraz programu Microsoft Visual Studio 2015

W tej początkowej części książki przedstawione zostaną kluczowe aspekty języka C# i zademonstrowane podstawowe techniki budowania aplikacji w Visual Studio 2015.

Z części I będzie się można dowiedzieć, jak tworzyć nowe projekty w Visual Studio oraz jak deklarować zmienne, wykorzystywać operatory do tworzenia wartości, wywoływać metody i pisać wiele instrukcji przydatnych w procesie implementowania programów C#. Będzie się można również dowiedzieć, jak obsługiwać wyjątki i stosować debuger Visual Studio do przechodzenia przez kod i wykrywania problemów, które mogą uniemożliwiać prawidłowe działanie aplikacji.

ROZDZIAŁ 1

Wprowadzenie do języka C#

Po ukończeniu tego rozdziału Czytelnik będzie potrafił:

- Korzystać ze środowiska programowania Microsoft Visual Studio 2015.
- Utworzyć aplikację konsolową w języku C#.
- Wyjaśnić cel stosowania przestrzeni nazw.
- Utworzyć prostą aplikację graficzną w języku C#.

Rozdział ten stanowi wprowadzenie do tematyki związanej z programem Visual Studio 2015, środowiskiem programowania oraz zestawem narzędzi stworzonych z myślą o ułatwieniu programiście tworzenia aplikacji przeznaczonych dla systemu Microsoft Windows. Program Visual Studio 2015 jest idealnym narzędziem do tworzenia kodu w języku C# i oferuje wiele różnorodnych funkcji, o których dowiemy się więcej w trakcie lektury tej książki. W tym rozdziale użyjemy programu Visual Studio 2015 do utworzenia kilku prostych aplikacji w języku C#, które będą stanowić wstęp do tworzenia wysoko funkcjonalnych rozwiązań dla systemu Windows.

Rozpoczynamy programowanie przy użyciu środowiska Visual Studio 2015

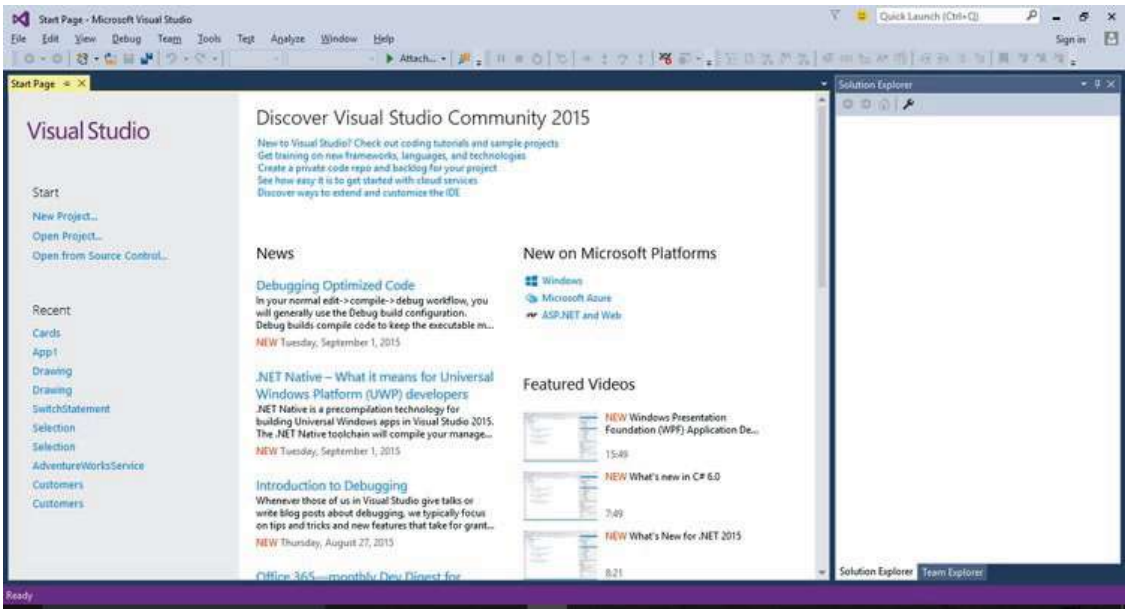
Visual Studio 2015 to rozbudowane środowisko programowania, oferujące funkcjonalność potrzebną przy tworzeniu zarówno małych, jak i dużych projektów w języku C#, przeznaczonych dla systemu Windows. Możliwe jest nawet konstruowanie projektów, które w gładki i bezproblemowy sposób łączą ze sobą moduły napisane przy użyciu różnych języków programowania, takich jak np. C++, Visual Basic lub F#. Nasze pierwsze ćwiczenie polegać będzie na uruchomieniu środowiska programowania Visual Studio 2015 i utworzeniu aplikacji konsolowej.

UWAGA Aplikacja konsolowa to aplikacja, która nie oferuje graficznego interfejsu użytkownika (GUI – ang. Graphical User Interface), lecz jest uruchamiana w oknie wiersza poleceń.

➜ Tworzenie aplikacji konsolowej przy użyciu Visual Studio 2015

1. W pasku zadań systemu Windows kliknij Start, wpisz **Visual Studio 2015**, a następnie naciśnij klawisz Enter.

Spowoduje to uruchomienie programu Visual Studio 2015 i wyświetlenie przez ten program strony Start, takiej jak ta pokazana poniżej (w zależności od używanej edycji programu Visual Studio 2015, strona **Start** na komputerze użytkownika może wyglądać nieco inaczej):



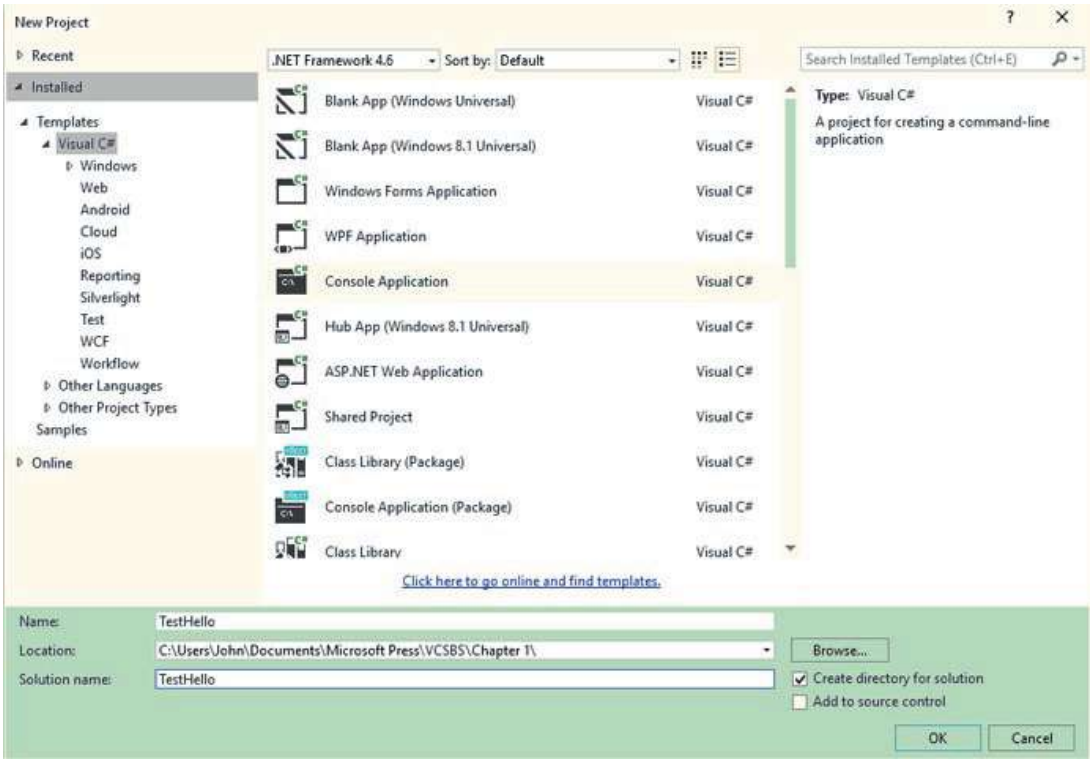
Uwaga Przy pierwszym uruchomieniu programu Visual Studio 2015 może zostać wyświetlone okno dialogowe umożliwiające wybór domyślnych ustawień środowiska programowania. Program Visual Studio 2015 dopasowuje się do preferowanego przez użytkownika języka programowania. Wybór języka programowania spowoduje zmianę domyślnych ustawień w różnych oknach dialogowych i narzędziach środowiska IDE (Integrated Development Environment – zintegrowane środowisko programowania). Należy wybrać z listy pozycję Visual C#, a następnie kliknąć przycisk Start Visual Studio (Uruchom Visual Studio). Spowoduje to wyświetlenie po chwili okna środowiska IDE Visual Studio 2015.

2. Wskaż w menu File (Plik) menu podrzędne New (Nowy), a następnie kliknij polecenie Project (Projekt).

   Spowoduje to otwarcie okna dialogowego New Project (Nowy projekt). Okno to zawierać będzie listę szablonów, które mogą zostać użyte jako punkt wyjściowy przy tworzeniu nowej aplikacji. Szablony widoczne w tym oknie dialogowym podzielone są na kategorie zależne od używanego języka programowania oraz rodzaju tworzonej aplikacji.

3. W panelu po lewej stronie rozwiń węzeł Installed (Zainstalowane), rozwiń węzeł Templates (Szablony), a następnie kliknij element Visual C#. Sprawdź, czy w polu

rozwijanej listy, znajdującym się w górnej części środkowego panelu, wybrana jest opcja .NET Framework 4.6, a następnie kliknij znajdującą się w tym panelu ikonę Console Application (Aplikacja konsolowa).



4. W polu Location (Lokalizacja) wpisz **C:\Users*TwojaNazwa*\Dokumenty\Microsoft Press\VCSBS\Chapter 1**. Występującą w tej ścieżce frazę *TwojaNazwa* zastąp własną nazwą użytkownika w systemie Windows.

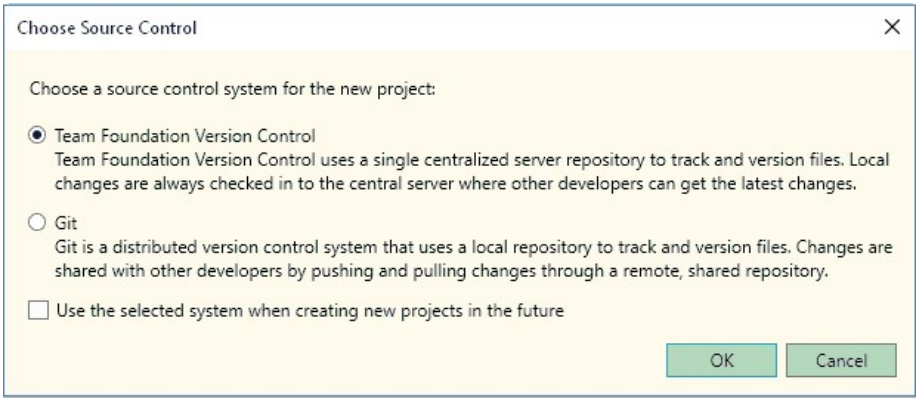
Uwaga Dla skrócenia zapisu, w dalszej części tej książki, zamiast odwoływać się do ścieżki C:\Users*TwojaNazwa*\Dokumenty, będziemy pisać po prostu o folderze Dokumenty użytkownika.

Wskazówka Jeśli określony folder nie istnieje, zostanie stworzony przez Visual Studio 2015.

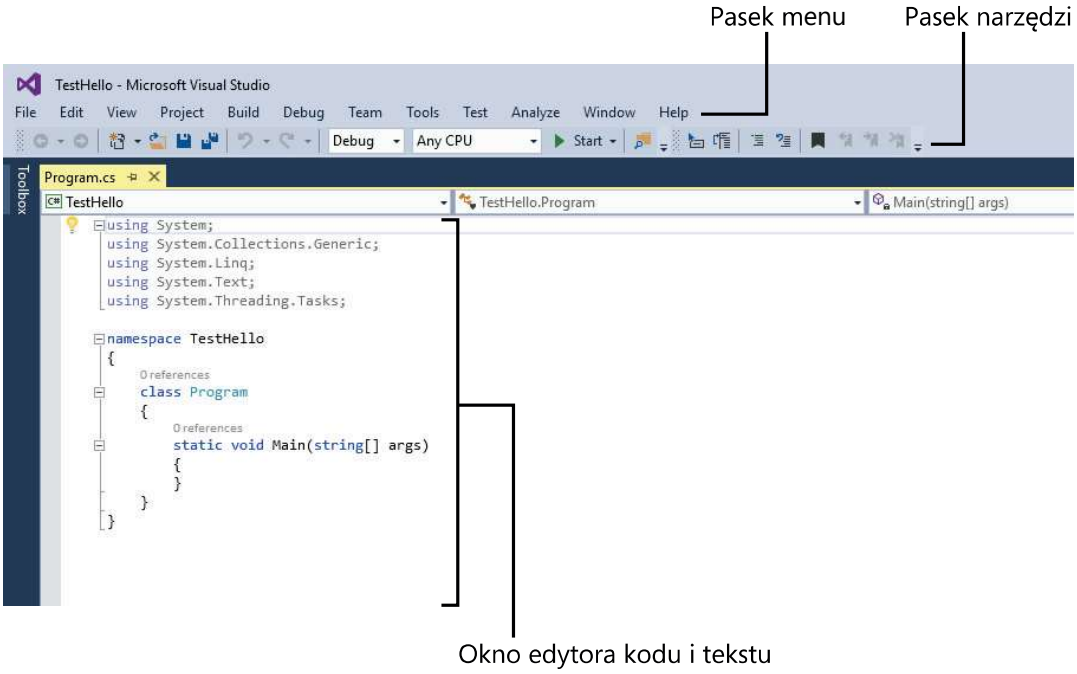
---

5. W polu Name (Nazwa) wpisz tekst **TestHello**, nadpisując znajdującą się w tym polu nazwę ConsoleApplication1.
6. Upewnij się, że pole wyboru opcji Create Directory for Solution (Utwórz katalog dla rozwiązania) jest zaznaczone oraz że pole wyboru Add To Source Control (Dodaj do kontroli źródła) jest odznaczone, a następnie kliknij przycisk OK.

Program Visual Studio utworzy nowy projekt korzystając z szablonu Console Application (Aplikacja konsolowa). Wyświetlenie następującego okna dialogowego monitującego o wybranie mechanizmu kontroli źródła do użycia oznacza, że pomyłkowo zaznaczone zostało pole wyboru Add To Source Control (Dodaj do kontroli źródła). W takiej sytuacji wystarczy kliknąć przycisk Cancel (Anuluj) i projekt zostanie utworzony bez kontroli źródła.



Visual Studio wyświetli początkowy kod aplikacji tak, jak to zostało pokazane na poniższym rysunku:

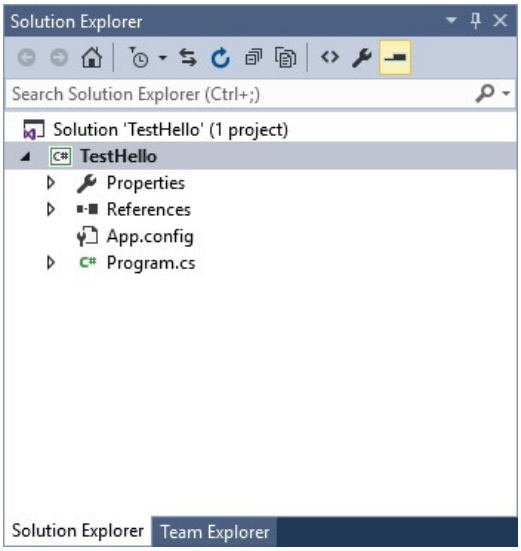


Pasek menu, znajdujący się w górnej części ekranu, zapewnia dostęp do różnych funkcjonalności używanych w środowisku programowania. Wszelkie polecenia oraz różne pozycje menu można uruchamiać przy pomocy klawiatury lub myszy, dokładnie w taki sam sposób, w jaki odbywa się to we wszystkich programach

opartych na systemie Windows. Poniżej paska menu znajduje się pasek narzędziowy, który zawiera przyciski będące skrótami umożliwiającymi uruchamianie najczęściej używanych komend.

W oknie edytora kodu i tekstu, zajmującym główną część ekranu, wyświetlana jest zawartość plików źródłowych. Jeśli podczas pracy z projektami złożonymi z kilku plików edytowany jest więcej niż jeden plik źródłowy, to każdy z tych plików posiadać będzie swoją własną zakładkę oznaczoną nazwą danego pliku. W celu przywołania wybranego pliku źródłowego na pierwszy plan okna edytora kodu i tekstu wystarczy kliknąć zakładkę z nazwą tego pliku.

Po prawej stronie okna wyświetlany jest panel Solution Explorer (Eksplorator rozwiązań), obok edytora kodu i tekstu:



W panelu Solution Explorer (Eksplorator rozwiązań) wyświetlane są między innymi nazwy związanych z projektem plików. Przywołanie wybranego pliku źródłowego na pierwszy plan okna edytora kodu i tekstu jest również możliwe poprzez dwukrotne kliknięcie tego pliku źródłowego w panelu eksploratora rozwiązań

Zanim przystąpimy do pisania kodu źródłowego, zapoznajmy się najpierw z plikami wyświetlanymi w panelu eksploratora rozwiązań, które zostały utworzone przez program Visual Studio 2015 jako część nowego projektu:

- **Solution (Rozwiązanie) 'TestHello'** Jest to plik leżący na najwyższym poziomie hierarchii rozwiązania. Każde rozwiązanie może zawierać jeden lub więcej projektów, a program Visual Studio 2015 tworzy plik rozwiązania, aby ułatwić organizowanie projektów. Jeśli sprawdzimy za pomocą Eksploratora zawartość swojego katalogu Dokumenty\Microsoft Press\VCSBS\Chapter 1\TestHello, to przekonamy się, że faktycznie plik ten nosi nazwę TestHello.sln.
- **TestHello** Jest to plik projektu w języku C#. Każdy plik projektu zawiera odwołania do jednego lub kilku plików zawierających kod źródłowy lub inne

elementy projektu, np. takie jak obrazy graficzne. Całość kodu źródłowego używanego w jednym projekcie musi być napisana w tym samym języku programowania. W programie Eksplorator Windows plik ten jest widoczny pod swoją faktyczną nazwą TestHello.csproj i znajduje się w katalogu użytkownika \Microsoft Press\VCSBS\Chapter 1\TestHello\TestHello.

- **Properties (Właściwości)** Jest to folder będący częścią projektu TestHello. Po rozwinięciu tego folderu (w tym celu należy kliknąć strzałkę znajdującą się obok nazwy Properties) przekonamy się, że zawiera on plik o nazwie AssemblyInfo.cs. Plik AssemblyInfo.cs to specjalny plik, który umożliwia dodawanie do programu różnych atrybutów, takich jak np. nazwa autora, data utworzenia programu itp. Możliwe jest także określenie dodatkowych atrybutów, zmieniających sposób działania programu. Omówienie sposobów korzystania z tych atrybutów wykracza jednak poza ramy tej książki.
- **References (Odwołania)** Ten folder zawiera odwołania do bibliotek skompilowanego kodu, które mogą być używane przez tworzoną aplikację. Podczas kompilacji kodu źródłowego napisanego w języku C# następuje konwersja tego kodu na bibliotekę, której zostanie nadana unikalna nazwa. W środowisku Microsoft .NET Framework biblioteki te nazywane są *zestawami wykonywalnymi* (ang. assembly). Programiści mogą używać zestawów wykonywalnych do umieszczania w nich napisanych przez siebie użytecznych fragmentów kodu w sposób umożliwiający ich dystrybuowanie i używanie przez innych programistów we własnych aplikacjach. Jeśli rozwiniemy folder References, to przekonamy się, że zawiera on zestaw domyślnych odwołań, dodanych do projektu przez program Visual Studio 2015. Te zestawy wykonywalne zapewniają dostęp do wielu powszechnie wykorzystywanych funkcji platformy .NET Framework i zostały dostarczone przez firmę Microsoft wraz z oprogramowaniem Visual Studio 2015. Podczas wykonywania zamieszczonych w tej książce ćwiczeń będziemy mieli okazję zapoznać się dokładniej z wieloma z tych zestawów wykonywalnych.
- **App.config** Jest to plik konfiguracyjny aplikacji. Plik ten jest plikiem opcjonalnym i nie zawsze musi występować. Plik konfiguracyjny umożliwia określanie ustawień, które będą mogły być wykorzystywane przez uruchomioną aplikację do modyfikowania sposobu jej działania, takich jak np. wersja platformy .NET Framework używana do uruchamiania danej aplikacji. Więcej informacji na temat tego pliku zostanie podanych w dalszych rozdziałach tej książki.
- **Program.cs** Jest to plik źródłowy w języku C#, który jest wyświetlany w oknie edytora kodu i tekstu bezpośrednio po utworzeniu projektu. W pliku tym będziemy zapisywać kod tworzonej aplikacji konsolowej. Plik ten zawiera także kod dodany do niego automatycznie przez program Visual Studio 2015, a który wkrótce dokładniej przeanalizujemy.

Piszemy pierwszy program

Plik Program.cs definiuje klasę o nazwie *Program*, która zawiera metodę o nazwie *Main*. W języku C# cały kod wykonywalny musi być zdefiniowany wewnątrz metody, a wszystkie metody muszą należeć do pewnej klasy lub ***struktury***. Więcej informacji na temat klas znajduje się w rozdziale 7, zatytułowanym „Tworzenie i zarządzanie klasami oraz obiektami", natomiast więcej informacji na temat struktur znaleźć można w rozdziale 9, zatytułowanym „Tworzenie typów wartości przy użyciu wyliczeń oraz struktur".

Metoda *Main* określa tzw. punkt wejścia do programu. Metoda ta musi być zdefiniowana w sposób określony w klasie *Program*, tj. jako metoda statyczna, gdyż w przeciwnym razie środowisko .NET Framework mogłoby nie rozpoznać tej metody jako punktu wejścia do programu. (Szczegóły budowy metod zostaną omówione w rozdziale 3, zatytułowanym „Tworzenie metod i stosowanie zasięgów zmiennych", a więcej informacji na temat metod statycznych znajduje się we wspomnianym już rozdziale 7).

Ważne W języku C# są rozróżniane małe i wielkie litery. Metoda *Main* musi mieć nazwę pisaną wielką literą.

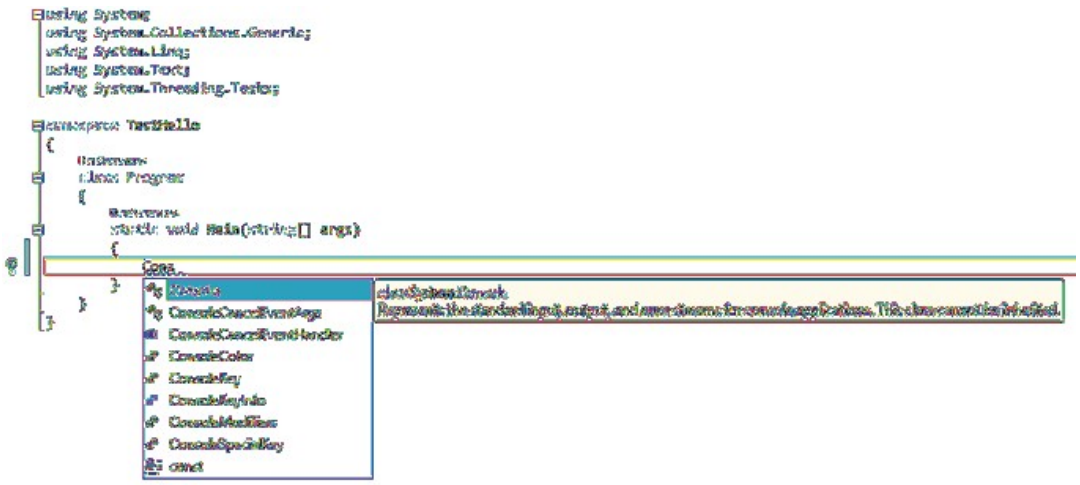
W kolejnych ćwiczeniach napiszemy kod wyświetlający w oknie konsoli komunikat „Hello World!" (Witaj świecie!); zbudujemy i uruchomimy naszą aplikację konsolową Hello World oraz poznamy sposób wykorzystywania przestrzeni nazw do dzielenia kodu na różne elementy.

➜ Pisanie kodu przy użyciu funkcji Microsoft IntelliSense

1. W oknie edytora kodu i tekstu, wyświetlającym zawartość pliku Program.cs, umieść kursor wewnątrz metody *Main*, bezpośrednio za otwierającym nawiasem klamrowym, {, a następnie naciśnij klawisz Enter, aby utworzyć nową linię.
2. W nowej linii wpisz słowo **Console**; jest to nazwa jeszcze jednej klasy zawartej w jednym z zestawów wykonywalnych dołączonych automatycznie do naszej aplikacji. Klasa ta oferuje metody pozwalające na wyświetlanie komunikatów w oknie konsoli oraz na odczytywanie danych wprowadzanych z klawiatury.

   Po wpisaniu litery **C**, będącej pierwszą literą słowa *Console*, wyświetlona zostanie lista funkcji IntelliSense.

   Lista ta zawierać będzie wszystkie słowa kluczowe języka C# oraz typy danych, które są poprawne w danym kontekście. Możesz albo kontynuować wpisywanie słowa, albo odszukać je na liście i dwukrotnie kliknąć myszą. Po wpisaniu liter **Cons** funkcja IntelliSense automatycznie podświetli na liście element *Console* i wówczas do jego wybrania wystarczy wciśnięcie klawisza Tab lub Enter.



Metoda *Main* powinna teraz wyglądać następująco:

```
static void Main(string[] args)
{
    Console
}
```

Uwaga Klasa *Console* jest klasą wbudowaną.

3. Wpisz znak kropki, bezpośrednio po słowie *Console*.

   Spowoduje to wyświetlenie nowej listy funkcji IntelliSense, na której wyświetlane będą nazwy metod, właściwości oraz pól klasy *Console*.

4. Przewiń dół zawartość tej listy, zaznacz na niej metodę *WriteLine*, a następnie wciśnij klawisz Enter. Możesz również wpisywać kolejne litery, **W**, **r**, **i**, **t**, **e**, **L**, aż do zaznaczenia na liście metody *WriteLine*, a następnie wcisnąć klawisz Enter.

   Okienko z listą funkcji IntelliSense zostanie wówczas zamknięte, a do pliku źródłowego zostanie dodane słowo *WriteLine*. Metoda *Main* powinna teraz wyglądać następująco:

```
static void Main(string[] args)
{
    Console.WriteLine
}
```

5. Wpisz znak nawiasu otwierającego, (. Spowoduje to wyświetlenie kolejnej podpowiedzi funkcji IntelliSense.

   Podpowiedź ta zawierać będzie listę parametrów akceptowanych przez metodę *WriteLine*. W rzeczywistości metoda *WriteLine* jest tzw. *metodą przeciążoną*, co oznacza, że klasa *Console* zawiera więcej niż jedną metodę o nazwie *WriteLine* – faktycznie klasa ta zawiera aż 19 różnych wersji tej metody. Każda z wersji metody

WriteLine pozwala na wyświetlanie na ekranie różnych typów danych. (Metody przeciążone zostaną omówione dokładniej w rozdziale 3). Metoda *Main* powinna teraz wyglądać następująco:

```
static void Main(string[] args)
{
    Console.WriteLine(
}
```

Wskazówka Klikanie znajdujących się w okienku podpowiedzi strzałek skierowanych w górę i w dół pozwala na przewijanie listy pomiędzy różnymi przeciążonymi wersjami metody *WriteLine*.

---

6. Wpisz znak nawiasu zamykającego,), a po nim znak średnika, ;.

   Metoda *Main* powinna teraz wyglądać następująco:

```
static void Main(string[] args)
{
    Console.WriteLine();
}
```

7. Przesuń kursor i wpisz pomiędzy znakami nawiasów występujących po nazwie metody *WriteLine*, tekst "**Hello World!**", włącznie ze znakami cudzysłowów.

   Metoda *Main* powinna teraz wyglądać następująco:

```
static void Main(string[] args)
{
    Console.WriteLine("Hello World!");
}
```

Wskazówka Warto wyrobić sobie nawyk, by znaki takie jak nawiasy zwykłe (i) oraz klamrowe { i } wpisywać parami, jeszcze przed wypełnieniem ich treścią. Jeśli odłożymy wpisanie znaku nawiasu zamykającego do chwili wprowadzenia treści, która powinna być ujęta w te nawiasy, można łatwo zapomnieć o wpisaniu znaku zamykającego.

Ikony funkcji IntelliSense

Po wpisaniu kropki po nazwie klasy funkcja IntelliSense wyświetla nazwy wszystkich elementów składowych tej klasy (jej członków). Z lewej strony nazwy każdego takiego elementu znajduje się ikona określająca jego rodzaj. Poniżej pokazane zostały typowe ikony wraz z ich znaczeniem:

Ciąg dalszy na stronie następnej

Ciąg dalszy ze strony poprzedniej

Ikona	Znaczenie
	Metoda (zostanie omówiona w rozdziale 3)
	Właściwość (zostanie omówiona w rozdziale 15)
	Klasa (zostanie omówiona w rozdziale 7)
	Struktura (zostanie omówiona w rozdziale 9)
	Zmienna wyliczeniowa (zostanie omówiona w rozdziale 9)
	Metoda rozszerzająca (zostanie omówiona w rozdziale 12)
	Interfejs (zostanie omówiony w rozdziale 13)
	Delegacja (zostanie omówiona w rozdziale 17)
	Zdarzenie (zostanie omówione w rozdziale 17)
	Przestrzeń nazw (zostanie omówiona w następnej części tego rozdziału)

Podczas wpisywania kodu w innym kontekście można spotkać się także z innymi ikonami funkcji IntelliSense.

W kodzie źródłowym często można spotkać linie zawierające dwa znaki ukośnika, //, po których następuje zwykły tekst. Są to komentarze – są one ignorowane przez kompilator, ale są bardzo użyteczne dla programistów, ponieważ ułatwiają dokumentowanie faktycznego sposobu działania programu. Przykładowo:

```
Console.ReadLine(); // Czekaj, aż użytkownik naciśnie klawisz Enter
```

Kompilator pomija cały tekst, począwszy od dwóch znaków ukośnika aż do końca danej linii. Możliwe jest także dodawanie komentarzy składających się z wielu linii, które rozpoczynają się od znaku ukośnika i gwiazdki – /*. Po napotkaniu tych znaków kompilator pomija wszystko, aż do napotkania sekwencji znaków gwiazdki i ukośnika */, która może znajdować się w pliku źródłowym nawet o wiele linii dalej. Zdecydowanie zachęcamy do dokumentowania własnego kodu źródłowego przy użyciu niezbędnej liczby wyczerpujących komentarzy.

➜ Budowanie i uruchamianie aplikacji konsolowej

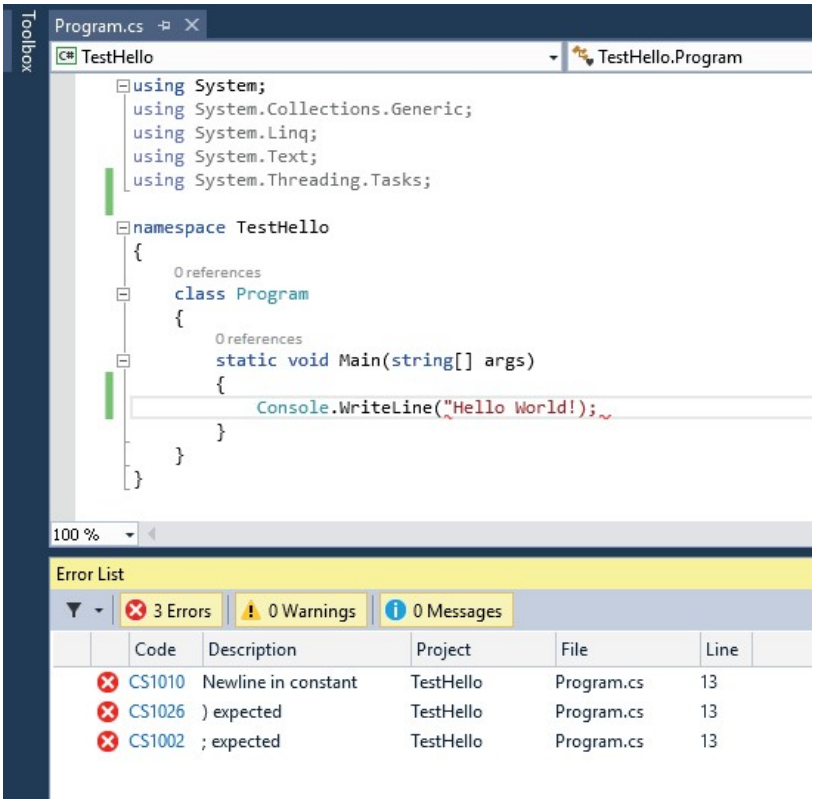
1. Wybierz z menu Build (Kompilowanie) polecenie Build Solution (Kompiluj rozwiązanie).

   Wykonanie tej akcji spowoduje skompilowanie kodu w języku C# i utworzenie wykonywalnego programu. Poniżej okna edytora kodu i tekstu wyświetlone zostanie okno Output (Dane wyjściowe).

Wskazówka Wykonanie tej akcji spowoduje skompilowanie kodu w języku C# i utworzenie wykonywalnego programu. Poniżej okna edytora kodu i tekstu wyświetlone zostanie okno Output (Dane wyjściowe).

W oknie Output (Dane wyjściowe) powinien wówczas zostać wyświetlony komunikat podobny do tego, który został pokazany poniżej, informujący o przebiegu procesu kompilacji programu:

```
1>------ Build started: Project: TestHello, Configuration: Debug Any CPU ------
1>  TestHello -> C:\Users\John\Dokumenty\Microsoft Press\Visual CSharp Step
By Step\Chapter
1\TestHello\TestHello\bin\Debug\TestHello.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```



Jeśli w kodzie źródłowym popełnione zostały jakieś błędy, to zostaną one wymienione w oknie Error List (Lista błędów). Zamieszczony na następnej stronie przykład pokazuje, co by się stało, gdybyśmy w instrukcji *WriteLine* zapomnieli wpisać zamykającego znaku cudzysłowu po tekście Hello World! Należy zwrócić uwagę na fakt, że czasami jedna pomyłka może prowadzić do wygenerowania kilku błędów kompilacji.

Wskazówka Dwukrotne kliknięcie wybranego elementu w oknie Error List (Lista błędów) spowoduje umieszczenie kursora w linii, która spowodowała dany błąd. Należy także zauważyć, że w kodzie źródłowym program Visual Studio podkreśla czerwoną falistą linią wszystkie wiersze, które nie będą mogły zostać poprawnie skompilowane.

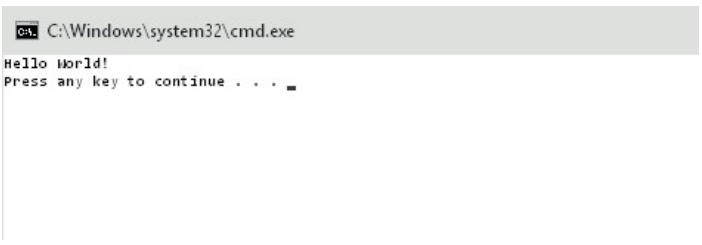
Jeśli wszystkie poprzednie instrukcje zostały wykonane dokładnie i z należytą starannością, to nie powinniśmy otrzymać żadnych błędów ani ostrzeżeń, a proces tworzenia programu powinien zakończyć się sukcesem.

Wskazówka Nie ma potrzeby jawnego zapisywania pliku źródłowego przed rozpoczęciem procesu budowy/kompilacji, ponieważ polecenie Build Solution (Kompiluj rozwiązanie) powoduje automatyczne zapisanie pliku źródłowego.

Gwiazdka widniejąca obok nazwy pliku na zakładce okna edytora kodu i tekstu oznacza, że dany plik został zmodyfikowany od czasu jego ostatniego zapisania na dysku.

2. Wybierz z menu Debug (Debugowanie) polecenie Start Without Debugging (Uruchom bez debugowania).

Spowoduje to otwarcie okna wiersza poleceń i uruchomienie programu. Uruchomiony program wyświetli komunikat Hello World! i będzie oczekiwać na wciśnięcie przez użytkownika dowolnego klawisza, tak jak to zostało pokazane na poniższym rysunku:

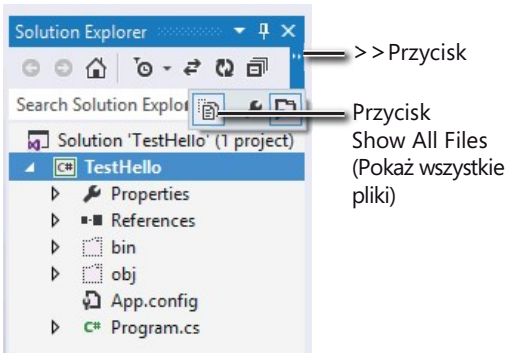


Uwaga Tekst monitu „Press any key to continue..." (Wciśnij dowolny klawisz, aby kontynuować..) został wygenerowany przez program Visual Studio; w naszym projekcie nie ma żadnego kodu powodującego wypisanie tego komunikatu. Jeśli program zostałby uruchomiony przy użyciu polecenia Start Debugging (Rozpocznij debugowanie) z menu Debug, to aplikacja również zostałaby uruchomiona, ale jej okno wyjściowe zostałoby natychmiast zamknięte, bez oczekiwania na wciśnięcie przez użytkownika dowolnego klawisza.

3. Upewnij się, że okno wiersza poleceń, w którym wyświetlane są rezultaty działania programu, jest aktywnym oknem (ma tzw. fokus), a następnie wciśnij klawisz Enter.

   Spowoduje to zamknięcie okna wiersza poleceń i powrót do środowiska programowania Visual Studio 2015.

4. W oknie Solution Explorer (Eksplorator rozwiązań) kliknij projekt *TestHello* (projekt, a nie rozwiązanie o tej samej nazwie), a następnie kliknij przycisk Show All Files (Pokaż wszystkie pliki) znajdujący się na pasku narzędziowym eksploratora rozwiązań. Należy pamiętać, że wyświetlenie tego przycisku może wymagać kliknięcia przycisku >>, znajdującego się po prawej stronie paska narzędziowego eksploratora rozwiązań



   Kliknięcie tego przycisku spowoduje wyświetlenie ponad plikiem *Program.cs* elementów o nazwach *bin* i *obj*. Elementy te odpowiadają folderom *bin* i *obj*, znajdującym się w folderze projektu (*Microsoft Press\Visual CSharp Step By Step\Chapter 1\TestHello\TestHello*). Foldery te są tworzone przez program Visual Studio podczas budowania aplikacji i zawierają wykonywalną wersję programu oraz pewne dodatkowe pliki, używane podczas procesu budowania aplikacji oraz podczas jej debugowania.

5. W oknie Solution Explorer (Eksplorator rozwiązań) rozwińgałąź *bin*.

   Ukaże się wówczas kolejny folder o nazwie Debug.

Uwaga W gałęzi tej może również znajdować się folder o nazwie Release.

6. Korzystając z okna Solution Explorer (Eksplorator rozwiązań) rozwiń folder Debug.

   Po rozwinięciu tego folderu pojawi się w nim kilka kolejnych elementów, wśród których znajdować się będzie plik o nazwie *TestHello.exe*. Plik ten to skompilowany program i to właśnie ten plik jest uruchamiany po wybraniu z menu Debug polecenia Start Without Debugging (Uruchom bez debugowania). Pozostałe trzy pliki zawierają informacje, które są używane przez program Visual Studio 2015 podczas uruchamiania programu w trybie debugowania – po wybraniu z menu Debug (Debugowanie) polecenia Start Debugging (Rozpocznij debugowanie).

Przestrzenie nazw

Prezentowany dotychczas przykład to bardzo mały program. Małe programy mogą jednak bardzo szybko rozrosnąć się do dużo większych rozmiarów. Wraz z powiększaniem się rozmiarów programu pojawiają się dwa główne problemy. Po pierwsze, w przypadku dużych programów utrzymywanie ich kodu oraz zrozumienie sposobu działania staje się trudniejsze niż w przypadku małych programów. Po drugie, większa ilość kodu zwykle oznacza większą liczbę klas, zawierających większą liczbę metod, to z kolei oznacza konieczność posługiwania się większą liczbą nazw. Wraz ze wzrostem liczby nazw rośnie również prawdopodobieństwo niepowodzenia procesu budowy projektu, spowodowane konfliktem dwóch lub więcej nazw; np. na skutek próby utworzenia dwóch klas o takiej samej nazwie. Sytuacja komplikuje się jeszcze bardziej, gdy tworzony program odwołuje się do zestawów wykonywalnych stworzonych przez innych programistów, którzy również posługują się wieloma różnymi nazwami.

W przeszłości programiści starali się rozwiązywać problem konfliktów nazw, poprzedzając je pewnego rodzaju kwalifikatorem (lub korzystając ze zbioru takich kwalifikatorów). Takie rozwiązanie nie było jednak najlepsze, ponieważ nie było skalowalne. Nazwy stawały się coraz dłuższe, a programiści spędzali coraz więcej czasu na wpisywaniu kodu, zamiast na jego pisaniu (to nie to samo) oraz na ciągłym odczytywaniu coraz bardziej niezrozumiałych nazw.

Przestrzenie nazw pomagają w rozwiązaniu tego problemu poprzez stworzenie kontenera dla innych identyfikatorów, takich jak np. nazwy klas. Dwie klasy o takiej samej nazwie nie zostaną ze sobą pomylone, jeśli będą należeć do różnych przestrzeni nazw. Przykładowo, utworzenie klasy *Pozdrowienia* w przestrzeni nazw *TestHello* przy użyciu słowa kluczowego *namespace* może wyglądać następująco:

```
namespace TestHello
{
   class Pozdrowienia
```

```
    {
      ...
    }
}
```

Do utworzonej w ten sposób klasy Pozdrowienia możemy odwoływać się w swoich programach przy użyciu nazwy *TestHello.Pozdrowienia*. Jeśli inny programista również utworzy klasę *Pozdrowienia* w innej przestrzeni nazw, np. w przestrzeni *NowaPrzestrzeńNazw* i zestaw wykonywalny zawierający tę klasę zostanie zainstalowany na naszym komputerze, to nasze programy nadal będą działać zgodnie z oczekiwaniami, ponieważ używają one klasy *TestHello.Pozdrowienia*. Jeśli zechcemy odwołać się do klasy *Pozdrowienia* utworzonej przez tego innego programistę, to będziemy musieli posłużyć się nazwą *NowaPrzestrzeńNazw.Pozdrowienia*.

Dobre praktyki programowania wymagają, aby wszystkie tworzone klasy były definiowane przy użyciu przestrzeni nazw, do której należą i środowisko Visual Studio 2015 stosuje się do tego zalecenia, używając nazwy projektu jako nazwy przestrzeni nazw najwyższego poziomu. Do zaleceń tych stosuje się również biblioteka klas platformy .NET Framework; każda zdefiniowana przez tę platformę klasa istnieje w pewnej przestrzeni nazw. Przykładowo, klasa *Console* istnieje w przestrzeni nazw *System*. Oznacza to, że faktyczna pełna nazwa tej klasy to *System.Console*.

Oczywiście, jeśli korzystając z klasy musielibyśmy za każdym razem wpisywać jej pełną nazwę, to sytuacja nie byłaby wcale lepsza niż wówczas, gdybyśmy stosowali jedynie jakąś formę przedrostków lub po prostu używali globalnie unikalnych nazw, takich jak np. *SystemConsole*. Na szczęście problem ten można rozwiązać stosując w swoich programach dyrektywę *using*. Jeśli cofniemy się do otwartego w środowisku Visual Studio 2015 projektu *TestHello* i przyjrzymy się widocznej w oknie edytora kodu i tekstu zawartości pliku *Program.cs*, to zauważymy na początku tego pliku następujące linie:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Linie te to dyrektywy *using*. Dyrektywa *using* powoduje włączenie wskazanej przestrzeni nazw do aktualnego zasięgu (ang. scope). W dalszej części kodu, znajdującej się w tym samym pliku źródłowym, nie ma już potrzeby jawnego kwalifikowania nazw obiektów nazwami przestrzeni nazw, z których pochodzą te obiekty. Pięć przestrzeni nazw pokazanych w tym przykładzie zawiera klasy, które są używane na tyle często, że program Visual Studio 2015 dodaje je automatycznie przy użyciu dyrektywy *using* do każdego nowo tworzonego projektu. Jeśli zachodzi potrzeba korzystania także z innych przestrzeni nazw, to oczywiście możliwe jest dodanie na początku pliku źródłowego kolejnych dyrektyw *using*.

Koncepcja przestrzeni nazw zostanie zaprezentowana dokładniej w poniższym ćwiczeniu.

➜ Ręczne wpisywanie długich nazw

1. W wyświetlanym w oknie edytora kodu i tekstu pliku *Program.cs* oznacz jako komentarz znajdującą się na początku pliku pierwszą dyrektywę *using*, tak jak to zostało pokazane poniżej:

   ```
   //using System;
   ```

2. Wybierz z menu Build (Kompilowanie) polecenie Build Solution (Kompiluj rozwiązanie).

   Proces kompilacji zakończy się niepowodzeniem, a w oknie Error List (Lista błędów) wyświetlony zostanie następujący komunikat błędu:

   ```
   The name 'Console' does not exist in the current context.
   (Nazwa ‘Console’ nie istnieje w bieżącym kontekście).
   ```

3. Kliknij dwukrotnie komunikat błędu wyświetlany w oknie Error List.

   Spowoduje to zaznaczenie w pliku źródłowym *Program.cs* identyfikatora, który spowodował wystąpienie tego błędu.

4. Popraw w oknie edytora kodu i tekstu treść metody *Main* tak, by używała ona w pełni kwalifikowanej nazwy metody *System.Console*.

   Metoda *Main* powinna wyglądać następująco:

   ```
   static void Main(string[] args)
   {
       System.Console.WriteLine("Hello World!");
   }
   ```

Uwaga Po wpisaniu znaku kropki po słowie *System*, funkcja IntelliSense wyświetli nazwy wszystkich elementów istniejących w przestrzeni nazw *System*.

5. Wybierz z menu Build (Kompilowanie) polecenie Build Solution (Kompiluj rozwiązanie).

   Tym razem proces budowania projektu zakończy się powodzeniem. Jeśli tak się nie stanie, to należy sprawdzić, czy metoda *Main* wygląda dokładnie tak jak w pokazanym wcześniej fragmencie kodu, a następnie ponowić próbę kompilacji kodu.

6. Uruchom aplikację, wybierając z menu Debug (Debugowanie) polecenie Start Without Debugging (Uruchom bez debugowania), aby przekonać się, że nadal działa ona poprawnie.

7. Po uruchomieniu programu i wypisaniu przez niego w oknie konsoli tekstu Hello World! wciśnij klawisz Enter, aby powrócić do programu Visual Studio 2015.

Przestrzenie nazw a wykonywalne zestawy binarne

Dyrektywa *using* powoduje po prostu włączenie elementów ze wskazanej przestrzeni nazw do bieżącego zasięgu (ang. scope), uwalniając programistę od konieczności używania w swoim kodzie w pełni kwalifikowanych nazw klas. Klasy są kompilowane w tzw. *binarne zestawy wykonywalne* (ang. assemblies). Binarny zestaw wykonywalny to plik, który zwykle ma rozszerzenie .dll, ale ściśle rzecz biorąc, są nimi również programy wykonywalne z rozszerzeniem .exe.

Binarny zestaw wykonywalny może zawierać wiele klas. Klasy składające się na bibliotekę klas platformy .NET Framework, takie jak np. *System.Console*, są dostarczane w zestawach binarnych instalowanych na komputerze razem z programem Visual Studio. Jak wkrótce się przekonamy, biblioteka klas platformy .NET Framework zawiera tysiące różnych klas. Gdyby wszystkie te klasy znajdowały się w jednym zestawie binarnym, miałby on bardzo duży rozmiar i był trudny do utrzymania (gdyby firma Microsoft uaktualniła tylko jedną metodę pojedynczej klasy, musiałaby na nowo rozdystrybuować całą bibliotekę klas do wszystkich programistów!).

Z tego względu biblioteka klas platformy .NET Framework została podzielona na kilka mniejszych zestawów wykonywalnych, odpowiadających różnym obszarom funkcjonalnym, z którymi związane są zawarte w nich klasy. Istnieje np. „zasadniczy" zestaw wykonywalny (zestaw ten nosi nazwę *mscorlib.dll*), który zawiera wszystkie typowe klasy, takie jak np. *System.Console*, a także inne zestawy wykonywalne, zawierające klasy służące do manipulowania bazami danych, korzystania z usług webowych, tworzenia graficznego interfejsu użytkownika itd. Jeśli zamierzamy skorzystać z klasy zawartej w binarnym zestawie wykonywalnym, konieczne jest dodanie we własnym projekcie odwołania do takiego zestawu. Następnie można dodać w kodzie źródłowym dyrektywę *using*, która spowoduje włączenie do zasięgu (ang. scope) elementów z przestrzeni nazw zawartych w danym zestawie binarnym.

Należy w tym miejscu podkreślić, że relacja pomiędzy binarnym zestawem wykonywalnym a przestrzenią nazw niekoniecznie musi być relacją typu 1:1. Pojedynczy zestaw wykonywalny może zawierać klasy zdefiniowane w wielu różnych przestrzeniach nazw, a pojedyncza przestrzeń nazw może rozciągać się na kilka zestawów wykonywalnych. Przykładowo, klasy oraz inne elementy z przestrzeni nazw *System* zostały faktycznie zaimplementowane w formie kilku różnych zestawów wykonywalnych, między innymi *mscorlib.dll*, *System.dll*, oraz

Ciąg dalszy na stronie następnej

System.Core.dll. Wszystko to może początkowo wydawać się bardzo zagmatwane, ale szybko można się do tego przyzwyczaić.

Szablon wybrany podczas tworzenia nowej aplikacji za pomocą programu Visual Studio powoduje automatyczne dołączenie odwołań do właściwych zestawów binarnych. Rozwińmy np. folder *References* (Odwołania), widoczny w oknie Solution Explorer (Eksplorator rozwiązań) dla otwartego projektu *TestHello*. Zobaczymy wówczas, że użycie szablonu *Console application* (Aplikacja konsolowa) spowodowało dołączenie odwołań do zestawów binarnych o nazwach: *Microsoft.CSharp, System, System.Core, System.Data, System.Data.DataSetExtensions, System.Net.Http, System.Xml* oraz *System.Xml.Linq*. Pewnym zaskoczeniem może być brak na tej liście zestawu *mscorlib.dll*. Wynika to z faktu, że zestaw ten zawiera pewne podstawowe funkcje i musi być używany przez wszystkie aplikacje korzystające z platformy .NET Framework. Folder *References* zawiera bowiem tylko opcjonalne zestawy wykonywalne i jeśli zachodzi taka potrzeba, to możliwe jest dodawanie nowych i usuwanie istniejących zestawów wykonywalnych z tego folderu.

Dodatkowe odwołania do zestawów wykonywalnych można dodać do projektu klikając prawym klawiszem myszy folder *References* i wybierając polecenie Add Reference (Dodaj odwołanie) – zadanie to zostanie wykonane podczas ćwiczeń zamieszczonych w dalszej części tego rozdziału. Operacja usunięcia zestawu wykonywalnego polega na kliknięciu prawym klawiszem myszy wybranego zestawu wyświetlanego w folderze *References*, a następnie wybraniu z menu kontekstowego polecenia Remove (Usuń).

Tworzenie aplikacji graficznej

Dotychczas użyliśmy programu Visual Studio 2015 do utworzenia i uruchomienia prostej aplikacji konsolowej. Środowisko programowania Visual Studio 2015 zawiera także wszystko, czego będziemy potrzebować do tworzenia graficznych aplikacji dla systemu Windows 10. Udostępnia szablony nazywane uniwersalnymi aplikacjami systemu Windows (Universal Windows Platform – UWP), ponieważ umożliwiają tworzenie aplikacji, które mogą być uruchamiane na dowolnym urządzeniu z systemem Windows, takim jak komputer, tablet czy telefon. Graficzny interfejs użytkownika dla systemu Windows można projektować w sposób interaktywny. Oprogramowanie Visual Studio 2015 wygeneruje następnie odpowiednie instrukcje programu, implementujące zaprojektowany interfejs użytkownika.

Środowisko Visual Studio 2015 oferuje dwa rodzaje widoków dla aplikacji graficznych: *widok projektowy* (Design view) oraz *widok kodu* (Code view). Okno edytora kodu i tekstu pozwala na modyfikowanie i utrzymywanie kodu oraz logiki tworzonej aplikacji graficznej, a widok projektowy pozwala na graficzne rozmieszczanie elementów

interfejsu użytkownika. Możliwe jest swobodne przełączenie się pomiędzy tymi dwoma widokami.

Zamieszczony poniżej zestaw ćwiczeńdemonstruje sposób tworzenia aplikacji graficznej przy użyciu Visual Studio 2015. Program ten wyświetlać będzie prosty formularz zawierający pole tekstowe, w którym można będzie wpisać swoje imię oraz przycisk służący do wyświetlania spersonalizowanego tekstu pozdrowienia w oknie komunikatu.

Więcej wskazówek oraz informacji dotyczących specyfiki procesu tworzenia aplikacji UWP znaleźć można w części IV tej książki.

➜ Tworzenie aplikacji graficznej w środowisku Visual Studio 2015

1. Uruchom program Visual Studio 2015, jeśli nie jest on jeszcze uruchomiony.
2. W menu File (Plik) wskaż menu podrzędne New (Nowy), a następnie wybierz z niego polecenie Project (Projekt).

   Spowoduje to otwarcie okna dialogowego New Project (Nowy projekt).
3. Rozwiń węzeł Installed (jeśli nie jest on już rozwinięty), rozwiń węzeł Templates (Szablony), rozwiń folder Visual C#, rozwińelement Windows, a następnie kliknij Universal.
4. Kliknij znajdującą się w środkowym panelu ikonę Blank App (Windows Universal) (Pusta aplikacja (aplikacja uniwersalna)).

Uwaga Skrót XAML oznacza Extensible Application Markup Language (Rozszerzalny język znaczników aplikacji). Język XAML jest używany przez aplikacje Universal Windows Platform do definiowania wyglądu graficznego interfejsu użytkownika. Więcej informacji na temat języka XAML zawierać będą opisy zamieszczonych w tej książce ćwiczeń.

---

5. Upewnij się, że ścieżka podana w polu Location (Lokalizacja) wskazuje katalog \Microsoft Press\VCSBS\Chapter 1 w folderze Dokumenty.
6. Wpisz w polu Name (Nazwa) tekst **Hello**.
7. Upewnij się, że zaznaczone zostało pole wyboru opcji Create Directory For Solution (Utwórz katalog dla rozwiązania), a następnie kliknij przycisk OK.

   Jeśli aplikacja UWP jest tworzona po raz pierwszy, to w tym miejscu może pojawić się monit o włączenie trybu dewelopera dla systemu Windows 10. Możliwość włączenia trybu dewelopera za pośrednictwem interfejsu użytkownika zależy od urządzenia oraz wersji systemu Windows 10. Wskazówki dotyczące włączania trybu dewelopera znaleźć można w artykule „Enable your device for development” dostępnym (w jęz. angielskim) pod adresem: *https://msdn.microsoft.com/library/windows/apps/xaml/dn706236.aspx*.

Uwaga To okno dialogowe może pojawić się podczas tworzenia nowej aplikacji UWP lub pierwszej próby uruchomienia aplikacji UWP w Visual Studio.

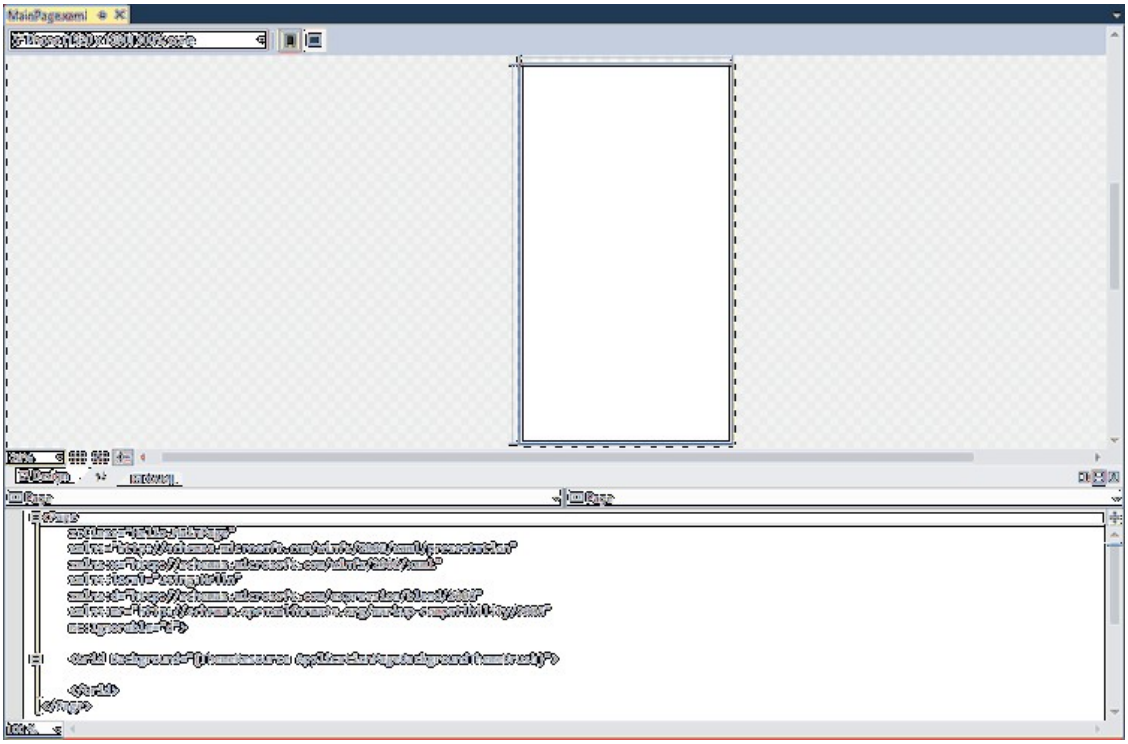
8. Po utworzeniu aplikacji przejrzyj zawartość okna Solution Explorer (Eksplorator rozwiązań).

   Nazwa użytego szablonu aplikacji jest nieco mylącą – wprawdzie Blank App oznacza „pustą aplikację", ale faktycznie szablon ten dostarcza kilku gotowych plików, zawierających całkiem pokaźną ilość kodu źródłowego. Jeśli na przykład otworzymy folder MainPage.xaml, znajdziemy w nim plik z kodem w języku C# o nazwie MainPage.xaml.cs. To właśnie w tym pliku zapisywany będzie nasz kod, który będzie wykonywany po wyświetleniu interfejsu użytkownika zdefiniowanego za pomocą pliku MainPage.xaml.

9. Korzystając z okna Solution Explorer kliknij dwukrotnie plik MainPage.xaml.

   Plik ten zawiera informacje o układzie interfejsu użytkownika. W oknie widoku projektowego wyświetlone zostaną dwie reprezentacje tego pliku (ilustracja na stronie następnej).

   W górnej części wyświetlany jest widok graficzny domyślnie odzwierciedlający wygląd 5-calowego ekranu telefonu, a w dolnej części znajduje się opis zawartości tego ekranu, zapisany w języku XAML. Język XAML jest podobny do języka XML i jest używany przez aplikacje UWP oraz aplikacje WPF do definiowania graficznego układu formularzy oraz ich zawartości. Dla osób znających język XML kod w języku XAML powinien wyglądać znajomo.



W następnym ćwiczeniu posłużymy się oknem widoku projektowego do rozmieszczenia elementów interfejsu użytkownika tworzonej aplikacji, a następnie zapoznamy się z wygenerowanym w ten sposób kodem w języku XAML.

Wskazówka Jeśli potrzebne będzie więcej miejsca do wyświetlania okna widoku projektowego, to można zamknąć okna Output (Dane wyjściowe) oraz Error List (Lista błędów).

Uwaga Zanim przejdziemy dalej, warto wyjaśnić pewne kwestie związane z używaną terminologią. W typowych aplikacjach Windows interfejs użytkownika składa się z jednego lub z kilku *okien*, ale w aplikacjach Universal Windows Platform elementy odpowiadające oknom nazywane są *stronami*. Dla uproszczenia oba te elementy będziemy nazywać po prostu ogólnym terminem *formularz*. Słowo *okno* nadal jednak będzie używane, ilekroć mowa będzie o elementach środowiska IDE oprogramowania Visual Studio 2015, takich jak np. okno widoku projektowego.

W kolejnych ćwiczeniach posłużymy się oknem widoku projektowego, za pomocą którego dodamy do wyświetlanego przez aplikację formularza trzy nowe kontrolki oraz zapoznamy się z kodem implementującym funkcjonalność tych kontrolek w języku C#, wygenerowanym automatycznie przez program Visual Studio 2015.

➜ Tworzenie interfejsu użytkownika

1. Kliknij zakładkę Toolbox (Przybornik), znajdującą się w oknie widoku projektowego po lewej stronie formularza.

   Spowoduje to wyświetlenie panelu narzędziowego Toolbox zawierającego różne komponenty oraz składniki, które można dodawać do tego formularza.

2. Rozwińsekcję o nazwie Common XAML Controls (Typowe kontrolki XAML).

   W sekcji tej znajduje się lista kontrolek używanych przez większość aplikacji graficznych.

Wskazówka Bardziej obszerną listę kontrolek można znaleźć w sekcji All XAML Controls (Wszystkie kontrolki XAML).

3. Kliknij ikonę kontrolki *TextBlock* (Blok tekstu), znajdującą się w sekcji Common XAML Controls, a następnie przeciągnij tę kontrolkę do formularza wyświetlanego w oknie widoku projektowego.

Wskazówka Należy koniecznie upewnić się, że zaznaczona została kontrolka *TextBlock* (Blok tekstu), a nie kontrolka *TextBox* (Pole tekstowe). Jeśli na formularzu została niechcący umieszczona niewłaściwa kontrolka, to można ją łatwo usunąć poprzez kliknięcie tej kontrolki i wciśnięcie klawisza Delete.

   Spowoduje to dodanie do formularza kontrolki *TextBlock* (za chwilę przeniesiemy ją we właściwe położenie) oraz ukrycie panelu Toolbox.

Wskazówka Jeśli chcesz, by panel Toolbox pozostał widoczny, lecz nie przesłaniał żadnej części formularza, kliknij przycisk Auto Hide (Autoukrywanie), znajdujący się po prawej stronie paska tytułowego panelu Toolbox. (Przycisk ten wygląda jak pinezka). Spowoduje to zakotwiczenie panelu Toolbox po lewej stronie okna programu Visual Studio 2015 oraz odpowiednie zmniejszenie rozmiarów okna widoku projektowego tak, by panel ten mógł zmieścić się na ekranie. (W przypadku korzystania z monitora o niskiej rozdzielczości może to oznaczać utratę bardzo dużego obszaru roboczego). Ponowne kliknięcie przycisku Auto Hide spowoduje ponowne ukrycie panelu Toolbox.

4. Umieszczona na formularzu kontrolka *TextBlock* prawdopodobnie nie znajduje się we właściwym miejscu. Położenie dodanych do formularza kontrolek możesz zmieniać poprzez ich kliknięcie i przeciągnięcie w żądane miejsce. Posługując się tą techniką przesuń kontrolkę *TextBlock* w okolicę górnego, lewego narożnika formularza. (W tym konkretnym przypadku dokładne umiejscowienie tej kontrolki

nie jest ważne). Należy pamiętać, że przesunięcie tej kontrolki może wymagać uprzedniego kliknięcia w inne miejsce okna widoku projektowego, a następnie ponownego kliknięcia tej samej kontrolki.

Opis formularza w języku XAML, który jest wyświetlany w dolnym panelu, zawiera teraz opis kontrolki *TextBlock*, wraz z jej właściwościami, takimi jak np. położenie kontrolki w obrębie formularza, określane przez właściwość *Margin* (Margines), tekst wyświetlany domyślnie przez kontrolkę, zapisany we właściwości *Text*, sposób wyrównywania tekstu wyświetlanego przez kontrolkę, określany przez właściwości *HorizontalAlignment* (Wyrównanie w poziomie) i *VerticalAlignment* (Wyrównanie w pionie) oraz to, czy tekst wykraczający poza szerokość kontrolki powinien być zawijany czy nie.

Kod w języku XAML dla kontrolki *TextBlock* będzie wyglądać podobnie jak w pokazanym poniżej przykładzie (konkretne wartości właściwości *Margin* mogą być nieco inne w zależności od miejsca, w którym umieszczona została na formularzu kontrolka *TextBlock*):

```
<TextBlock x:Name="textBlock" HorizontalAlignment="Left" Margin="10,10,0,0"
TextWrapping="Wrap"
Text="TextBlock" VerticalAlignment="Top"/>
```

Panel kodu w języku XAML oraz okno widoku projektowego są ze sobą nawzajem powiązane. Możliwe jest edytowanie wartości w panelu XAML, a wszelkie wykonane w tym panelu zmiany zostaną odzwierciedlone w oknie widoku projektowego. Możliwa jest na przykład zmiana położenia kontrolki *TextBlock* poprzez odpowiednią modyfikację wartości zapisanych we właściwości *Margin*.

5. Wybierz z menu View (Widok) polecenie Properties Window (Okno właściwości).

Polecenie to spowoduje wyświetlenie okna Properties (Właściwości) w dolnej prawej części ekranu, poniżej panelu Solution Explorer (Eksplorator rozwiązań), o ile okno to nie było wyświetlane już wcześniej. Właściwości kontrolek można określać także za pomocą panelu XAML, znajdującego się poniżej okna z widokiem projektowym. Okno Properties (Właściwości) oferuje jednak wygodniejszy sposób modyfikowania właściwości umieszczonych na formularzu elementów, a także innych elementów projektu.

Zawartość okna Properties zależna jest od kontekstu, co oznacza, że wyświetlane są w nim właściwości aktualnie zaznaczonego elementu. Kliknięcie w oknie widoku projektowego dowolnego miejsca formularza poza kontrolką *TextBlock* spowoduje wyświetlenie w oknie Properties właściwości elementu typu *Grid* (Siatka). Jeśli sprawdzimy zawartość panelu XAML, to przekonamy się, że kontrolka *TextBlock* znajduje się wewnątrz elementu typu *Grid*. Wszystkie formularze zawierają element typu *Grid*, który zawiera rozkład wyświetlanych elementów; np.