

Spis treści

<i>Podziękowania</i>	xv
<i>O autorze</i>	xvii
<i>Wstęp</i>	xix

Część I Wprowadzenie do języka Visual C# i Microsoft Visual Studio 2022

1 Wprowadzenie do języka C#	3
Pisanie pierwszego programu w języku C#	3
Zaczynamy programować w środowisku Visual Studio 2022	10
Pisanie pierwszego programu przy użyciu Visual Studio 2022	15
Przestrzenie nazw	23
Przestrzenie nazw i asemblacje	25
Komentowanie kodu	26
Tworzenie aplikacji graficznej	27
Analiza aplikacji Universal Windows Platform	40
Dodawanie kodu do aplikacji graficznej	44
Podsumowanie	49
Krótki przegląd rozdziału 1	49
2 Zmienne, operatory i wyrażenia	51
Instrukcje	51
Identyfikatory	52
Identyfikowanie słów kluczowych	53
Zmienne	54
Nazywanie zmiennych	54
Deklarowanie zmiennych	55
Specyfikowanie wartości numerycznych	56
Podstawowe typy danych	57
Zmienne lokalne bez przypisanej wartości	57
Wyświetlanie wartości podstawowych typów danych	58
Posługiwanie się operatorami arytmetycznymi	66
Operatory i typy danych	66
Poznajemy operatory arytmetyczne	69
Kontrolowanie pierwszeństwa	75
Stosowanie zasad łączności przy wyznaczaniu wartości wyrażen	76
Zasady łączności a operator przypisania	77

Inkrementacja i dekrementacja wartości zmiennych.....	78
Formy przyrostkowe i przedrostkowe	79
Deklarowanie zmiennych lokalnych o niejawnie określonym typie danych....	80
Podsumowanie	81
Krótki przegląd rozdziału 2	82
3 Tworzenie metod i stosowanie zakresów zmiennych.....	83
Tworzenie metod	83
Deklarowanie metody	84
Zwracanie danych przez metodę.....	85
Stosowanie metod wcielających wyrażenie	86
Wywoływanie metod.....	88
Składnia wywołania metody	88
Zwracanie wielu wartości z metody	91
Stosowanie zakresu	93
Definiowanie zakresu lokalnego.....	94
Definiowanie zakresu klasy	95
Przeciążanie metod	96
Tworzenie metod	97
Używanie debugera Visual Studio do krokowego wykonywania metod.....	102
Refaktoryzowanie kodu	110
Zagnieżdżanie metod	111
Stosowanie parametrów opcjonalnych i nazwanych argumentów.....	114
Definiowanie parametrów opcjonalnych.....	116
Przekazywanie nazwanych argumentów.....	117
Rozwiązywanie niejednoznaczności związanych z parametrami opcjonalnymi i argumentami nazwanymi	117
Podsumowanie	124
Krótki przegląd rozdziału 3	124
4 Instrukcje wyboru.....	127
Deklarowanie zmiennych logicznych.....	128
Stosowanie operatorów logicznych	128
Operatory równościowe oraz operatory relacji.....	129
Warunkowe operatory logiczne.....	129
Skracanie działania.....	130
Podsumowanie informacji o pierwszeństwie oraz łączności operatorów ..	131
Dopasowywanie wzorców	132
Podejmowanie decyzji przy użyciu instrukcji if	133
Składnia instrukcji if	133
Grupowanie instrukcji w bloki.....	134
Kaskadowe łączenie instrukcji if	136

Stosowanie instrukcji switch	142
Składnia instrukcji switch	143
Reguły stosowania instrukcji switch	144
Używanie wyrażeń switch z dopasowywaniem wzorców	149
Podsumowanie	152
Krótki przegląd rozdziału 4	152
5 Złożone instrukcje przypisania oraz instrukcje iteracji	155
Złożone operatory przypisania	155
Instrukcja while	157
Instrukcja for	163
Instrukcja do	166
Podsumowanie	176
Krótki przegląd rozdziału 5	176
6 Obsługa błędów i wyjątków	177
Próbowanie kodu i przechwytywanie wyjątków	178
Nieobsłużone wyjątki	180
Stosowanie kilku bloków obsługi	181
Przechwytywanie wielu wyjątków	182
Filtrowanie wyjątków	184
Propagowanie wyjątków	189
Wykonywanie operacji arytmetycznych z kontrolą lub bez kontroli	
przepełnienia	191
Pisanie instrukcji z kontrolą przepełnienia	192
Pisanie wyrażeń z kontrolą przepełnienia	192
Zgłaszanie wyjątków	196
Używanie wyrażeń throw	202
Stosowanie bloku finally	203
Podsumowanie	204
Krótki przegląd rozdziału 6	205

Część II Model obiektowy języka C#

7 Tworzenie i zarządzanie klasami oraz obiektami	209
Istota klasyfikacji	209
Cele hermetyzacji	210
Definiowanie i używanie klas	210
Kontrolowanie dostępności	213
Konstruktory	215
Przeciążanie konstruktorów	216
Dekonstrukcja obiektu	226

Metody i dane statyczne	227
Tworzenie pól współdzielonych	228
Tworzenie pól statycznych przy użyciu słowa kluczowego const	229
Klasy statyczne	229
Statyczne instrukcje using	230
Klasy anonimowe	233
Podsumowanie	235
Krótki przegląd rozdziału 7	235
8 Wartości i referencje	237
Kopiowanie klas oraz zmiennych typu wartościowego	237
Wartości null oraz typy danych dopuszczające stosowanie wartości null.	245
Operatory warunkowe wartości null	246
Nullowalne typy danych	248
Właściwości typów nullowalnych	249
Używanie parametrów typu ref i out	250
Tworzenie parametrów typu ref	251
Tworzenie parametrów typu out	252
Organizacja pamięci komputera	254
Korzystanie ze stosu i ze sterty	256
Klasa System.Object	257
Opakowywanie typów danych	258
Rozpakowywanie typów danych	259
Bezpieczne rzutowanie danych	261
Operator is	261
Operator as	262
Ponowne odwiedziny instrukcji switch	262
Podsumowanie	266
Krótki przegląd rozdziału 8	267
9 Tworzenie typów wartościowych przy użyciu wyliczeń oraz struktur	269
Wyliczeniowe typy danych	269
Deklarowanie wyliczeniowego typu danych	270
Stosowanie wyliczeniowych typów danych	270
Wybór wartości literałów wyliczeniowych	271
Wybór typu danych używanego do wewnętrznego reprezentowania wartości wyliczeniowych	272
Struktury	275
Deklarowanie struktury	277
Różnice pomiędzy strukturami i klasami	278
Deklarowanie zmiennych strukturalnych	280

Inicjowanie struktur	280
Kopiowanie zmiennych strukturalnych	285
Podsumowanie	289
Krótki przegląd rozdziału 9	289
10 Tablice	291
Deklarowanie i tworzenie tablicy	292
Tworzenie instancji tablicy	292
Wypełnianie tablic danymi i ich używanie	294
Tworzenie tablic o niejawnie określonym typie elementów	295
Uzyskiwanie dostępu do elementów tablicy	296
Uzyskiwanie dostępu do serii elementów tablicy	297
Wykonywanie iteracji poprzez elementy tablicy	297
Przekazywanie tablic jako parametrów i zwracanie ich jako wartości metod	299
Kopiowanie tablic	301
Tablice wielowymiarowe	303
Tworzenie tablic nieregularnych	304
Dostęp do tablic zawierających typy wartościowe	315
Podsumowanie	319
Krótki przegląd rozdziału 10	319
11 Tablice parametrów	321
Przeciążanie: krótkie przypomnienie faktów	321
Używanie argumentów tablicowych	322
Deklarowanie tablicy params	324
Używanie parametru typu params object[]	326
Stosowanie tablicy parametrów typu params	328
Porównanie tablic parametrów z parametrami opcjonalnymi	331
Podsumowanie	334
Krótki przegląd rozdziału 11	334
12 Dziedziczenie	335
Czym jest dziedziczenie?	335
Używanie dziedziczenia	336
Powtórka informacji na temat klasy System.Object	338
Wywoływanie konstruktora klasy bazowej	339
Przypisywanie klas	340
Deklarowanie nowych metod	342
Deklarowanie metod wirtualnych	343
Deklarowanie metod przesłaniających	345
Dostęp chroniony	348

Tworzenie metod rozszerzających.....	354
Podsumowanie	359
Krótki przegląd rozdziału 12	359
13 Tworzenie interfejsów oraz definiowanie klas abstrakcyjnych	361
Interfejsy	361
Definiowanie interfejsu	363
Implementowanie interfejsu	363
Odwoływanie się do klasy za pomocą jej interfejsu	365
Praca z wieloma interfejsami.....	366
Jawne implementowanie interfejsu.....	367
Obsługa wersjonowania a interfejsy	369
Ograniczenia interfejsów	371
Definiowanie i używanie interfejsów.....	371
Klasy abstrakcyjne	381
Metody abstrakcyjne	383
Klasy zamknięte.....	384
Metody zamknięte	384
Implementowanie i używanie klas abstrakcyjnych	385
Podsumowanie	392
Krótki przegląd rozdziału 13	393
14 Oczyszczanie pamięci i zarządzanie zasobami.....	395
Żywot obiektów.....	395
Tworzenie finalizatorów	397
Dlaczego trzeba używać kolektorów śmieci?.....	400
Działanie procesu oczyszczania pamięci	402
Zalecenia	403
Zarządzanie zasobami.....	404
Metody sprząające	404
Sprzątanie w sposób odporny na występowanie wyjątków.....	405
Instrukcja using oraz interfejs IDisposable	406
Wywoływanie metody Dispose z poziomu finalizatora.....	407
Implementacja metody sprząającej odpornej na występowanie wyjątków ..	410
Obsługiwanie zwalniania asynchronicznego	420
Podsumowanie	421
Krótki przegląd rozdziału 14	422

Część III Rozszerzalne typy danych w języku C#

15	Implementacja właściwości zapewniających dostęp do pól	427
	Implementacja hermetyzacji przy użyciu metod	428
	Czym są właściwości?	430
	Używanie właściwości	433
	Właściwości tylko do odczytu	433
	Właściwości tylko do zapisu	434
	Dostępność właściwości	434
	Ograniczenia właściwości	435
	Deklarowanie właściwości interfejsu	437
	Zastępowanie metod właściwościami	439
	Dopasowywanie wzorców we właściwościach	443
	Generowanie automatycznych właściwości	444
	Inicjowanie obiektów przy użyciu właściwości	446
	Automatyczne właściwości i niezmiennosc	448
	Używanie rekordów z właściwościami do implementowania lekkich struktur	451
	Podsumowanie	457
	Krótki przegląd rozdziału 15	457
16	Indeksatory i obsługa danych binarnych	461
	Czym jest indeksator?	461
	Przechowywanie danych binarnych	462
	Wyświetlanie wartości dwójkowych	463
	Manipulowanie wartościami dwójkowymi	463
	Ten sam przykład z wykorzystaniem indeksatorów	465
	Akcesory indeksatora	467
	Porównanie indeksatorów i tablic	468
	Indeksatory w interfejsach	470
	Stosowanie indeksatorów w aplikacjach Windows	471
	Podsumowanie	478
	Krótki przegląd rozdziału 16	478
17	Typy generyczne	481
	Problem niewłaściwego użycia typu object	481
	Rozwiązanie z użyciem typów generycznych	485
	Typy generyczne a klasy uogólnione	488
	Typy generyczne i nakładanie ograniczeń	488
	Tworzenie klasy generycznej	488
	Teoria drzew binarnych	489
	Budowanie klasy drzewa binarnego przy użyciu typu generycznego	492
	Tworzenie metody generycznej	502

Definiowanie metody generycznej do budowy drzewa binarnego	503
Wariancja i interfejsy generyczne	508
Interfejsy kowariantne	510
Interfejsy kontrawariantne	511
Podsumowanie	514
Krótki przegląd rozdziału 17	514
18 Kolekcje	517
Czym są klasy kolekcji?	517
Klasa kolekcji List<T>	519
Klasa kolekcji LinkedList<T>	522
Klasa kolekcji Queue<T>	524
Klasa kolekcji PriorityQueue<TElement, TPriority>	525
Klasa kolekcji Stack<T>	526
Klasa kolekcji Dictionary<TKey, TValue>	527
Klasa kolekcji SortedList<TKey, TValue>	528
Klasa kolekcji HashSet<T>	529
Używanie inicjalizatorów kolekcji	531
Metody Find, predykaty i wyrażenia lambda	532
Różne formy wyrażen lambda	534
Wyrażenia lambda i metody anonimowe	536
Porównanie tablic i kolekcji	536
Podsumowanie	541
Krótki przegląd rozdziału 18	542
19 Wyliczanie kolekcji	545
Wyliczanie elementów kolekcji	545
Ręczna implementacja modułu wyliczającego	547
Implementowanie interfejsu IEnumerable	552
Implementowanie modułu wyliczającego przy użyciu iteratora	554
Prosty iterator	555
Definiowanie modułu wyliczającego dla klasy Tree<TItem> przy użyciu iteratora	557
Podsumowanie	559
Krótki przegląd rozdziału 19	560
20 Wydzielanie logiki aplikacji i obsługa zdarzeń	561
Czym są delegaty	562
Przykłady delegatów w bibliotece klas .NET	563
Scenariusz zautomatyzowanej fabryki	564
Deklarowanie i używanie delegatów	569
Delegaty i wyrażenia lambda	578

Włączanie powiadomień za pomocą zdarzeń	580
Deklarowanie zdarzenia	580
Subskrypcja zdarzenia	581
Anulowanie subskrypcji zdarzenia	582
Zgłaszanie zdarzenia	582
Zdarzenia interfejsu użytkownika	583
Używanie zdarzeń	584
Podsumowanie	590
Krótki przegląd rozdziału 20	591
21 Odpytywanie danych w pamięci przy użyciu wyrażeń w języku zapytań	593
Czym jest LINQ?	594
Używanie LINQ w aplikacjach C#	594
Wybieranie danych	596
Filtrowanie danych	599
Porządkowanie, grupowanie i agregowanie danych	600
Złączanie danych	602
Operatory zapytań	603
Odpytywanie danych w obiektach <code>Tree<TItem></code>	606
LINQ i opóźnione przetwarzanie	612
Podsumowanie	615
Krótki przegląd rozdziału 21	616
22 Przeciążanie operatorów	619
Czym są operatory	619
Ograniczenia operatorów	620
Operatory przeciążone	621
Tworzenie operatorów symetrycznych	622
Przetwarzanie złożonych instrukcji przypisania	624
Deklarowanie operatorów inkrementujących i dekrementujących	624
Operatory porównań w strukturach i klasach	625
Definiowanie par operatorów	626
Implementowanie operatorów	627
Przesłanie operatorów równościowych	631
Operatory konwersji	634
Wbudowane metody konwersji	634
Implementowanie własnych operatorów konwersji	635
Tworzenie operatorów symetrycznych – uzupełnienie	636
Tworzenie operatorów konwersji	637
Podsumowanie	639
Krótki przegląd rozdziału 22	640

Część IV Tworzenie aplikacji Universal Windows Platform w języku C#

23	Przyspieszanie działania za pomocą zadań	643
	Dlaczego realizować wielozadaniowość przy użyciu przetwarzania równoległego?	643
	Narodziny procesora wielordzeniowego	644
	Implementowanie wielozadaniowości w .NET	646
	Zadania, wątki i pula wątków	646
	Tworzenie, uruchamianie i kontrolowanie zadań	648
	Używanie klasy Task do implementacji równoległości	651
	Tworzenie abstrakcji zadań za pomocą klasy Parallel	661
	Kiedy nie używać klasy Parallel	666
	Anulowanie zadań i obsługa wyjątków	668
	Mechanizm anulowania kooperatywnego	669
	Obsługiwanie wyjątków zadań za pomocą klasy AggregateException	681
	Kontynuowanie w przypadku zadań anulowanych lub przerwanych z powodu wyjątku	683
	Podsumowanie	684
	Krótki przegląd rozdziału 23	684
24	Skracanie czasu reakcji za pomocą działań asynchronicznych	687
	Implementowanie metod asynchronicznych	688
	Definiowanie metod asynchronicznych: problem	689
	Definiowanie metod asynchronicznych: rozwiązanie	692
	Definiowanie metod asynchronicznych zwracających wartości	698
	Pułapki metod asynchronicznych	700
	Metody asynchroniczne i interfejsy API środowiska Windows Runtime	701
	Zadania, alokacje pamięci i wydajność	704
	Używanie PLINQ do zrównoleglania deklaratywnego dostępu do danych	707
	Wykorzystanie PLINQ do poprawy wydajności przy wykonywaniu iteracji po elementach kolekcji	708
	Anulowanie zapytania PLINQ	713
	Synchronizowanie współbieżnych operacji dostępu do danych	714
	Blokowanie danych	717
	Elementarne narzędzia synchronizacji umożliwiające koordynowanie zadań	718
	Anulowanie synchronizacji	721
	Współbieżne klasy kolekcji	721
	Wykorzystanie kolekcji współbieżnej i blokady do implementacji dostępu do danych przystosowanego do trybu wielowątkowego	722

Podsumowanie	733
Krótki przegląd rozdziału 24.....	734
25 Implementowanie interfejsu użytkownika aplikacji	
Universal Windows Platform.....	737
Cechy aplikacji Universal Windows Platform	739
Budowa aplikacji UWP przy użyciu szablonu Blank App	742
Implementowanie skalowalnego interfejsu użytkownika.....	746
Implementowanie układu tabelarycznego za pomocą kontrolki Grid	757
Dostosowywanie układu za pomocą menedżera stanów wizualnych	766
Stosowanie stylów do interfejsu użytkownika	774
Podsumowanie	785
Krótki przegląd rozdziału 25.....	785
26 Wyświetlanie i wyszukiwanie danych w aplikacjach	
Universal Windows Platform.....	787
Implementowanie wzorca projektowego Model-View-ViewModel	787
Wyświetlanie danych przy użyciu wiązania	788
Modyfikowanie danych przy użyciu wiązania danych.....	795
Stosowanie wiązania danych do kontrolki ComboBox	800
Tworzenie składnika ViewModel	802
Dodawanie poleceń do składnika ViewModel.....	807
Podsumowanie	817
Krótki przegląd rozdziału 26.....	818
27 Dostęp do zdalnej bazy danych z poziomu aplikacji	
Universal Windows Platform.....	819
Pobieranie informacji z bazy danych	819
Tworzenie modelu encji	830
Tworzenie i korzystanie z usługi web typu REST.....	837
Aktualizowanie aplikacji UWP, aby używała usługi web	853
Wyszukiwanie danych w aplikacji Customers	863
Wstawianie, aktualizacja i usuwanie danych za pośrednictwem	
usługi web typu REST.....	870
Podsumowanie	872
Krótki przegląd rozdziału 27.....	873
Indeks.....	875

Podziękowania

Witajcie ponownie! Zapraszam do 10 wydania. W podziękowaniach do poprzednich wydań wspominałem o malowaniu Forth Railway Bridge i o Syzyfie jako nigdy niekończących się zadaniach. Być może w przyszłości ktoś doda do tej legendarnej listy pracę polegającą na aktualizowaniu Microsoft C# Krok po kroku. Co powiedziawszy, pisanie i uaktualnianie książki jest jednak znacznie bardziej satysfakcjonujące, niż oskrobywanie rdzy lub codzienne wtaczanie kamienia na szczyt, a dodatkową zaletą jest to, że pewnego dnia będę mógł po prostu przejść na emeryturę.

Pomimo faktu, że na okładce 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.

Po pierwsze, Loretta Yates z Pearson Education, która przyjęła rolę nadzorcy popychającego mnie do działania, i choć uprzejmie, to bezwzględnie przypominała mi dobrze zdefiniowane terminy realizacji poszczególnych etapów. Bez jej energii i uporu projekt ten nigdy by nie wystartował.

Następnie Rick Kughen, niestrudzony redaktor, który zadbał o to, aby moja pisa- nina była choć połowicznie zrozumiała, wyszukując brakujące słowa i bezsensowne frazy w tekście.

Dalej wkracza Charvi Arora i jej niestrudzony zespół redakcyjny, a w szczególności Kate Shoup i Dan Foster, którzy zadbali o to, aby moja gramatyka była przynajmniej akceptowalna i wyłapali wszystkie pominięte słowa lub nonsensowne frazy w tekście. Następnie David Fransen, który miał zadanie przejrzenia i przetestowania kodu przykładów i ćwiczeń. Wiem z własnego doświadczenia, że jest to niewdzięczne i frustrujące zadanie, ale poświęcony czas i niezliczone uwagi są jedynym sposobem, aby książka stała się lepsza. Oczywiście dowolne błędy, które pozostały, są wyłącznie moją winą i będę szczęśliwy, gdy czytelnicy powiadomią mnie o tych, które dostrzegą.

Jak zwykle muszę podziękować Dianie, mojej lepszej połowie, która dbała o stałe dostawy gorących napojów (złożonych głównie z kofeiny), gdy zbliżały się końcowe terminy. W czasie lockdownu Covid-19 uznała, że nasz dom nie jest dostatecznie wypełniony, zatem wprowadziła do naszej rodziny dwa raczej szalone kociaki. Psy są teraz w ciągłym strachu, ale mamy w zamian nieskończone godziny zabaw z podnoszeniem i opuszczaniem zasłon i graniem w „polowanie na mysz/żabę/pajaka” lub cokolwiek innego, co udało im się złapać i przynieść do domu. Nie wiem, jak mogłem do tej pory żyć bez nich.

Wreszcie na koniec muszę wspomnieć o naszych dzieciach, Jamesie i Frankie, którzy oboje już wyfrunęli z rodzinnego gniazda. James spędził ostatnich kilka lat w Manili, pracując (jak twierdzi) dla rządu brytyjskiego. Sądząc po fotografiach, wygląda to raczej na niekończące się wakacje na plażach południowo-wschodniej Azji. Frankie pozostała bliżej domu, dzięki czemu od czasu do czasu wpada i bierze udział w łapaniu myszy/żab/pajaków. Przy okazji uwaga do tych deweloperów, którymi kieruje w swojej pracy – najwyższy czas, abyście zrobili jej filiżankę herbaty!

O autorze

JOHN SHARP jest głównym technologiem w CM Group Ltd. części Civica Group, brytyjskiej firmy programistycznej i konsultingowej. Jest szeroko znanym programistą, konsultantem, projektantem i autorem wielu materiałów szkoleniowych, poradników i książek. Jego przeszło 35-letnie doświadczenie zawodowe obejmuje szeroki zakres różnych technologii, poczynając od programowania w Pascalu dla systemów CP/M i tworzenia aplikacji C/Oracle na różnych wersjach systemów UNIX, aż po projektowanie rozproszonych aplikacji w C# i JavaScript i programowanie dla Windows 11 i Microsoft Azure. Swój czas poświęca również na tworzenie kursów szkoleniowych dla firmy Microsoft, koncentrując się na takich obszarach, jak danetyka z wykorzystaniem języków R i Python, przetwarzanie Big Data przy użyciu Spark i CosmosDB, SQL Server i NoSQL, usługi web, Blazor, a także programowanie międzyplatformowe przy użyciu takich frameworków, jak Xamarin i MAUI oraz skalowalne architektury aplikacji oparte na Azure.

Wstęp

Wiele zmieniło się w ciągu ostatnich 20 lat. Czasami dla poprawy nastroju sięgam po swój egzemplarz pierwszego wydania *Microsoft C# Krok po kroku*, który ukazał się w 2001 roku i wzdycham wyrozumiale nad swoją naiwnością i niewinnością w tamtym czasie. Bez wątpienia C# stanowił w tym czasie szczyt perfekcji wśród języków programowania. C# oraz .NET Framework przebojem zdobyły świat deweloperów i echa tych wydarzeń rozbrzmiewają do tej pory. Jednak nie wygasają z czasem, ale mają coraz większy wpływ na powstawanie oprogramowania. Zamiast podejścia jednoplatformowego, co krytykowali przeciwnicy .NET w roku 2001, C# i .NET stają się stopniowo kompletnym, wieloplatformowym rozwiązaniem, które pozwala budować aplikacje dla Windows, macOS, Linuksa lub Androida. Dodatkowo C# i .NET pokazały już, że są wybierane jako środowisko wykonawcze wielu systemów chmurowych. Czymże Azure byłoby bez nich?

W przeszłości większość powszechnych języków programowania od czasu do czasu przechodziło okazjonalne aktualizacje, często rozciągające się na wiele lat. Dla przykładu, jeśli spojrzymy na Fortran, zauważymy standardy o nazwach Fortran 66, Fortran 77, Fortran 90, Fortran 95, Fortran 2003, Fortran 2008 oraz Fortran 2018. Mamy więc siedem aktualizacji w ciągu 55 lat. Choć taki względnie powolny cykl zmian promuje stabilność, prowadzi również do stagnacji. Problem leży w tym, że natura problemów, z którymi deweloperzy muszą sobie radzić, zmienia się błyskawicznie, zaś narzędzia, na których się opierają, powinny dotrzymywać kroku tym zmianom, aby możliwe było tworzenie efektywnych rozwiązań. Microsoft .NET zapewnia ciągle ewoluujący framework, zaś C# podlega częstym uaktualnieniom, aby zapewnić najlepsze wykorzystanie platformy. Tak więc w przeciwieństwie do Fortranu, C# podlegał bardzo szybkiej ewolucji od pierwszego wydania – sześć wersji w ostatnich pięciu latach, przy czym kolejna zmiana jest zaplanowana na rok 2022. Język C# nadal wspiera kod napisany ponad 20 lat temu, ale dzisiejsze dodatki i rozszerzenia języka pozwalają tworzyć rozwiązania używające bardziej eleganckiego kodu i bardziej zwięzłych konstrukcji. Z tych powodów dokonuję okresowych aktualizacji tej książki i tak doszliśmy do bieżącego, dziesiątego wydania!

Poniższa lista przedstawia krótką historię C#:

- C# 1.0 miał swój publiczny debiut w roku 2001.
- C# 2.0 wraz z programem Visual Studio 2005 udostępnił kilka ważnych i nowych funkcji, takich jak ogólne typy wyliczeniowe i metody anonimowe.

- C# 3.0, która pojawiła się wraz z opublikowaniem programu Visual Studio 2008, dodał obsługę metod rozszerzających, wyrażeń lambda 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 *późne wiązanie* (*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 wykorzystywać możliwości wielordzeniowych procesorów.
- W C# 5.0 dodano natywną obsługę dla przetwarzania zadań w sposób asynchroniczny poprzez użycie modyfikatora metody `async` oraz operatora `await`.
- C# 6.0 była kolejnym ulepszeniem zaprojektowanym 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.
- W C# 7.0 do 7.3 pojawiły się dalsze ulepszenia poprawiające produktywność i usuwające pewne anachronizmy. Na przykład od tych wersji możemy implementować akcesory właściwości jako elementy wcielające wyrażenia, metody mogą zwracać wiele wartości (w formie krotek), uproszczone zostało użycie parametrów `out`, a instrukcje `switch` zostały rozszerzone o obsługę dopasowywania wzorców i typów. Te wersje języka zawierały również wiele innych, drobnieszych poprawek, rozwiązujących problemy zgłaszane przez deweloperów, takie jak dopuszczenie asynchroniczności metody `Main`.
- C# 8.0, C# 9.0 i C# 10.0 kontynuują te ulepszanie języka w celu poprawy czytelności kodu i zwiększenia produktywności programistów. Niektóre z głównych uzupełnień obejmują rekordy, których możemy używać do budowania niezmiennych typów referencyjnych; rozszerzenia dopasowywania wzorców, pozwalające na używanie tej funkcjonalności w całym języku, a nie jedynie w instrukcjach `switch`; instrukcje najwyższego poziomu, które pozwalają używać C# jako języka skryptowego (nie musimy już zawsze pisać metody `Main`); domyślne metody interfejsu; statyczne funkcje lokalne; asynchroniczne typy odrzucalne i wiele innych funkcji omówionych w tej książce.

Nie da się zaprzeczyć, że Microsoft Windows jest ważną platformą dla aplikacji tworzonych w C#, ale teraz możemy uruchamiać kod zaprojektowany w C# również

w innych systemach operacyjnych, takich jak Linux, dzięki środowisku .NET Runtime. Otwiera to możliwości tworzenia kodu, który może działać w wielu środowiskach. Dodatkowo Windows wspiera aplikacje o wysokim stopniu interaktywności, które mogą współdzielić pomiędzy sobą dane i współpracować ze sobą nawzajem lub łączyć się z usługami działającymi w chmurze. Kluczowym pojęciem w Windows są aplikacje Universal Windows Platform (UWP) – zaprojektowane tak, aby mogły być uruchamiane na dowolnym urządzeniu Windows 10 lub 11, począwszy od 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.

Chmura stała się tak ważnym elementem architektury wielu systemów – od wielkoskalowych aplikacji dla przedsiębiorstw, po apki mobilne działające na urządzeniach przenośnych – że zdecydowałem skupić się na tym aspekcie wytwarzania oprogramowania w finalnym rozdziale książki.

Środowisko programowania oferowane przez pakiet Visual Studio 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 wykorzystaniu środowiska Visual Studio 2022 oraz platformy .NET Framework w wersji 6 lub późniejszej. 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.

Dla kogo nie jest przeznaczona ta książka

Niniejsza książka skierowana jest do osób, które dopiero uczą się języka C#, ale nie są nowicuszami 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, Azure lub Windows Communication Foundation. Czytelnicy oczekujący większej ilości informacji na temat jednej z tych technologii powinni rozważyć lekturę kilku innych tytułów wydanych przez Microsoft Press.

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
Programistą początkującym w dziedzinie programowania obiektowego	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 od 1 do 22. 3. Przeczytaj rozdziały 23 do 27 w miarę podnoszenia poziomu doświadczenia oraz potrzeb.
Programistą dobrze obeznanym z proceduralnymi językami programowania, takimi jak np. język C, ale nowicjuszem w 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 2022, a następnie skoncentruj się na rozdziałach od 6 do 22. 3. Przeczytaj rozdziały 23 do 27 w miarę podnoszenia poziomu doświadczenia oraz potrzeb.
Programistą mającym już doświadczenie w innych obiektowych 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 2022, a następnie skoncentruj się na rozdziałach od 7 do 22. 3. Przeczytaj rozdziały 23 do 27 w miarę podnoszenia poziomu doświadczenia oraz potrzeb.

Jeżeli jesteś	Wykonaj następujące kroki
Programistą znającym język Visual Basic, 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 od 1 do 22. 3. Przeczytaj rozdziały 23 do 27 w miarę podnoszenia poziomu doświadczenia oraz potrzeb. 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 2022.
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 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 szeregu czynności 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”, „Wskazówka”, zawierają dodatkowe informacje lub opis alternatywnego sposobu wykonania danego kroku.
- Tekst, który powinien zostać wpisany przez Czytelnika, wyróżniany jest wytłuszczoną czcionką.
- Znak plus pomiędzy oznaczeniem dwóch klawiszy oznacza, że należy klawisze te nacisnąć jednocześnie, np. Alt+Tab oznacza, że trzeba przytrzymać klawisz Alt w chwili naciśnięcia klawisza Tab.

Wymagania systemowe

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

- System operacyjny Windows 10 albo Windows 11 (Home, Professional, Education lub Enterprise).
- Najnowsze wydanie Visual Studio Community 2022, Visual Studio Professional 2022 lub Visual Studio Enterprise 2022*. Jako minimum należy podczas instalacji wybrać następujące środowiska:
 - Projektowanie dla Universal Windows Platform
 - Projektowanie .NET
 - Projektowanie ASP.NET i Web
 - Projektowanie Azure
 - Przechowywanie i przetwarzanie danych
 - Projektowanie międzyplatformowe .NET Core



UWAGA Wszystkie ćwiczenia i przykłady kodu z tej książki zostały opracowane i prze-testowane przy użyciu Visual Studio Community 2022. Powinny wszystkie działać bez modyfikacji w Visual Studio Professional 2022 oraz Visual Studio Enterprise 2022.

- Komputer z 64-bitowym procesorem 1.8 GHz lub szybszym, zalecany procesor czterordzeniowy lub lepszy. Procesory ARM nie są obsługiwane.
- 4 GB pamięci RAM (zalecane 8 GB lub więcej).
- Miejsce na dysku twardym: od 850 MB w konfiguracji minimalnej do 210 GB wolnego miejsca, zależnie od zainstalowanych funkcji. Typowa instalacja wymaga od 20 do 50 GB.
- Karta graficzna obsługująca rozdzielczość minimum 720p (1280 × 720). Visual Studio najlepiej działa w rozdzielczości WXGA (1366x768) lub wyższej.

* Podczas przygotowywania polskiego wydania książki przyjęliśmy założenie, że Czytelnik posługuje się systemem Windows w polskiej wersji językowej i że używa Visual Studio z zainstalowanym polskim pakietem językowym. Nazwy poleceń menu i elementów programu występujące w tekście są podawane w wersji polskiej – przy pierwszym użyciu w nawiasie podawana jest nazwa angielska. Natomiast kod przykładowy opracowany przez autora nie był modyfikowany – przetłumaczone zostały jedynie komentarze występujące w tym kodzie. Wszystkie zrzuty ekranowe zostały wykonane na komputerze systemu Windows 10 w Visual Studio Community 2022 wersja 17.2.6. Oprogramowanie Visual Studio podlega ciągłemu rozwojowi, zatem nie jest wykluczone, że w chwili, gdy książka trafi do rąk Czytelników, pewne elementy interfejsu użytkownika będą wyglądały inaczej, niż w tej publikacji (przyp. red. wydania polskiego).

- 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 2022 może wymagać posiadania uprawnień lokalnego administratora.

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

Przykłady kodu

Większość rozdziałów tej książki zawiera 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) oraz w wersji końcowej (tj. takiej, jaką otrzymalibyśmy po starannym wykonaniu całego ćwiczenia) z następującej strony:

<https://MicrosoftPressStore.com/VisualCsharp10e/downloads>

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 z powyższej strony web do swojego katalogu Dokumenty.
2. Zaakceptuj postanowienia licencyjne w wyświetlonym monicie.

UWAGA Jeśli tekst umowy licencyjnej nie zostanie wyświetlony, z treścią tej umowy można zapoznać się na tej samej stronie web, z której pobrany został plik z przykładowymi kodami źródłowymi.



Korzystanie z przykładowych kodów źródłowych

Każdy rozdział tej książki wyjaśnia, kiedy i jak należy korzystać z przykładowych kodów źródłowych dla danego rozdziału. Gdy nadejdzie pora użycia kolejnego przykładu, podane zostaną dokładne instrukcje, jak należy otworzyć odpowiednie pliki.



WAŻNE Wiele przykładowych projektów jest zależnych od pakietów NuGet, które nie zostały dołączone do przykładów kodu. Te pakiety zostaną 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 2022, 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. Finalne wersje projektów z każdego rozdziału znajdują się w folderach o nazwie uzupełnionej przyrostkiem „- Complete” (Gotowe).

Projekt	Opis
Chapter 1	
HelloWorld	Projekt ten pozwala rozpocząć naukę. Prowadzi Czytelnika poprzez tworzenie prostego programu za pomocą edytora tekstu (Notatnika). Program ten wyświetla tekstowe pozdrowienia.
HelloWorld2	Ten projekt demonstruje wykorzystanie interfejsu CLI (Command Level Interface) środowiska .NET do zbudowania i uruchomienia prostej aplikacji C#.
TextHello	Projekt Visual Studio wyświetlający tekst pozdrowienia.
HelloUWP	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	W tym projekcie ponownie przeanalizujemy kod poprzedniego projektu MathsOperators i zbadamy wykorzystanie metod do uporządkowania kodu źródłowego.
DailyRate	Projekt demonstrujący proces pisania własnych metod, ich uruchamiania oraz krokowego śledzenia przy pomocy debugera z programu Visual Studio 2022.

Projekt	Opis
DailyRate Using Optional Parameters	Projekt demonstrujący sposób definiowania metod akceptujących parametry opcjonalne oraz wywoływania tych metod przy użyciu nazwanych argumentów.
Factorial	Projekt ten demonstruje metodę rekurencyjną obliczającą silnię.
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.
SwitchStatement using Pattern Matching	Jest to rozszerzona wersja projektu SwitchStatement, która wykorzystuje dopasowywanie wzorców w celu uproszczenia logiki instrukcji switch.
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.
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, jak różne nieobsłużone wyjątki mogą doprowadzić do przerwania działania programu. Stabilność działania aplikacji zostanie 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.

Projekt	Opis
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.
Chapter 13	
Drawing	Projekt implementujący część pakietu graficznego. Projekt ten wykorzystuje interfejsy do zdefiniowania metod udostępnianych i implementowanych przez różne figury geometryczne.
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 opracowaną 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.
Student enrollment	Projekt demonstrujący użycie rekordów do modelowania strukturalnych typów niezmiennych.
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.

Projekt	Opis
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 modelowania puli kart rozdanych pomiędzy graczy w grze karcianej.
Chapter 19	
BinaryTree	Projekt demonstrujący sposób zaimplementowania ogólnego interfejsu typu <code>IEnumerator<T></code> , który umożliwia utworzenie obiektu wyliczeniowego dla ogólnej klasy <code>Tree</code> .
IteratorBinaryTree	Rozwiązanie wykorzystujące iterator do wygenerowania typu wyliczeniowego dla ogólnej klasy <code>Tree</code> .
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 zespolone i implementujący kilka typowych operatorów dla tego typu.
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 <code>GraphDemo</code> , w którym do wyodrębnienia procesu tworzenia i zarządzania zadaniami użyta została klasa <code>Parallel</code> .

Projekt	Opis
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.
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 <code>async</code> oraz operator <code>await</code> .
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.
ParallelTest	Program ilustrujący zagrożenia wynikające z pozwalania na niekontrolowany dostęp do współdzielonych danych przez równoległe wątki.
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 <i>INotifyPropertyChanged</i> , 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.

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.
Customers with insert and update features	To rozwiązanie zawiera uaktualnioną wersję projektu Customers, wykorzystującego usługę web typu REST do tworzenia rekordów nowych klientów i modyfikowania szczegółów istniejących klientów.

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:

<https://MicrosoftPressStore.com/VisualCsharp10e/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. Jeśli ktoś potrzebuje pomocy dotyczącej oprogramowania lub sprzętu firmy Microsoft, powinien skorzystać z informacji na poniższej stronie:

<https://support.microsoft.com>

CZĘŚĆ I

Wprowadzenie do języka Visual C# i Microsoft Visual Studio 2022

1	Wprowadzenie do języka C#	2
2	Zmienne, operatory i wyrażenia	51
3	Tworzenie metod i stosowanie zakresów zmiennych	83
4	Instrukcje wyboru	127
5	Złożone instrukcje przypisania oraz instrukcje iteracji	155
6	Obsługa błędów i wyjątków	177

Wstępna część książki przedstawia kluczowe aspekty języka C# i demonstruje podstawowe techniki budowania aplikacji w Visual Studio 2022.

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 debugger 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:

- Utworzyć aplikację konsolową w języku C#.
- Korzystać ze środowiska programowania Microsoft Visual Studio 2022.
- Wyjaśnić cel stosowania przestrzeni nazw.
- Utworzyć prostą aplikację graficzną w języku C#.

Rozdział ten stanowi wprowadzenie do Visual Studio 2022, środowiska programowania oraz zestawu narzędzi stworzonych z myślą o ułatwieniu programiście tworzenia aplikacji przeznaczonych dla systemu Microsoft Windows. Program Visual Studio 2022 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 Visual Studio 2022 do utworzenia kilku prostych aplikacji w języku C#, które będą stanowić wstęp do tworzenia wysoko funkcjonalnych rozwiązań dla systemu Windows.

Pisanie pierwszego programu w języku C#

Najprostszy sposób rozpoczęcia nauki niemal dowolnego języka programowania jest napisanie wszechobecnej aplikacji „Hello World!” – możliwie najprostszego programu działającego w konsoli komputera i wyświetlającego komunikat „Hello World!”

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



Visual Studio 2022 udostępnia graficzne, interaktywne środowisko programowania (*interactive development environment* – IDE). Zawiera ono wyrafinowane edytory, funkcje sprawdzania składni, zarządzania plikami i narzędzia organizacji projektu, a także inne elementy. Pozwala uzyskać wysoką produktywność, ale na początku bezmiar opcji i poleceń może być nieco przytłaczający. Dla zachowania prostoty (początkowo) w pierwszych kilku ćwiczeniach w tej książce będziemy pracować w trybie wiersza poleceń systemu Windows i użyjemy zestawu narzędzi wiersza poleceń .NET

(*command-line interface* – CLI). Narzędzia te są instalowane jako część Visual Studio 2022. Nasze pierwsze programy w C# napiszemy za pomocą Notatnika. Gdy już utworzymy parę prostych aplikacji i uzyskamy podstawową wiedzę, jak to wszystko działa, przełączymy się na IDE Visual Studio.

Tworzenie i uruchamianie aplikacji Hello World!

1. W pasku zadań Windows zaznacz pole **Start**, wpisz **cmd**, po czym naciśnij klawisz **Enter**.
2. W oknie wiersza poleceń wpisz **cd C:\Users\<TwojaNazwa>\Documents\Microsoft Press\VCSBS\Chapter 1**, aby przejść do folderu zawierającego kod dla rozdziału 1. Zastąp tekst <TwojaNazwa> swoją nazwą użytkownika systemu Windows.



UWAGA Aby uniknąć powtórzeń i oszczędzić miejsce, w całej książce będę się odwoływać do ścieżki C:\Users\TwojaNazwa\Documents po prostu jako do folderu *Documents**.

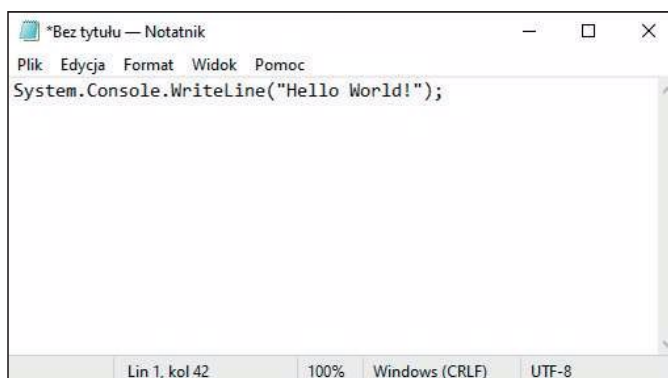
3. Uruchom poniższe polecenie, aby utworzyć nowy folder o nazwie HelloWorld:

```
mkdir HelloWorld
```

4. Wpisz **Notatnik**, po czym naciśnij **Enter**.

5. W Notatniku wpisz poniższy tekst:

```
System.Console.WriteLine("Hello World!");
```



Ten wiersz kodu wywołuje funkcję `WriteLine`, która wypisuje łańcuch znaków wyspecyfikowany jako argument (wartość umieszczoną w nawiasach) we wskazanym miejscu docelowym. Funkcja `WriteLine` należy do klasy o nazwie `Console`, która odpowiada ekranowi (oknu wiersza poleceń). Dowolny tekst przekazany do metody `WriteLine` klasy `Console` zostanie wypisany na ekranie. Klasa `Console`

* Użytkownicy polskiej wersji Microsoft Windows w interfejsie graficznym będą widzieć nazwę „Dokumenty”, ale rzeczywista nazwa katalogu na dysku to *Documents*.

należy do biblioteki obiektów dostarczanych wraz z C#. Elementy tej biblioteki są zebrane w przestrzeni nazw `System`. Ta przestrzeń nazw i odpowiadająca jej biblioteka zawiera wiele klas narzędziowych, pozwalających na wykonywanie fundamentalnych operacji w języku C#. Wiele z tych klas zbadamy w trakcie lektury tej książki.

UWAGA Mówiąc precyzyjnie, funkcje należące do klas, takie jak `WriteLine`, powinny być nazywane *metodami*. Będziemy poznawać metody począwszy od rozdziału 3, „Tworzenie metod i stosowanie zakresów zmiennych”.



6. Otwórz menu **Plik**, wybierz **Zapisz jako**, po czym zapisz plik jako **Program.cs** w podfolderze **Microsoft Press\VCSBS\Chapter 1\HelloWorld** folderu **Documents**.

UWAGA Wszystkie pliki kodu w języku C# powinny mieć rozszerzenie nazwy `.cs`.



7. Otwórz menu **Plik** i wybierz **Nowy**, aby utworzyć nowy, pusty plik.
8. Dodaj do tego pliku poniższy tekst:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>
</Project>
```

Jest to przykład *pliku projektu C#*. Narzędzia CLI .NET wykorzystają ten plik do skompilowania naszego kodu w aplikację wykonywalną. Zawartość tego pliku określa typ pliku wykonywalnego, który ma zostać utworzony (plik EXE) oraz wersję środowiska wykonawczego .NET (*runtime*), którego należy użyć do zbudowania i uruchomienia aplikacji (.NET 6.0). W dalszej części książki zobaczymy, jak tworzyć inne pliki wykonywalne, takie jak biblioteki łączone dynamicznie (*dynamic-link library* – DLL), które można współużytkować w wielu aplikacjach.

9. Otwórz menu **Plik**, wybierz **Zapisz jako** i zapisz plik pod nazwą **Hello.csproj** w podfolderze **Microsoft Press\VCSBS\Chapter 1\HelloWorld** w swoim folderze **Documents**.

UWAGA Pliki projektów C# powinny mieć rozszerzenie nazwy `.csproj`.



10. Zamknij Notatnik, powróć do okna wiersza poleceń, po czym wpisz poniższe polecenie, aby przejść do folderu `HelloWorld`:

```
cd HelloWorld
```

11. Wywołaj polecenie **dir** i upewnij się, że folder zawiera dwa dopiero co utworzone pliki: HelloWorld.csproj oraz Program.cs.

```
Wiersz polecenia
C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld>dir
Volume in drive C is Windows-SSD
Volume Serial Number is 18F7-0F50

Directory of C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld

03.08.2022  15:46    <DIR>          .
03.08.2022  15:46    <DIR>          ..
18.01.2022  19:33                167 Hello.csproj
18.01.2022  19:33                41 Program.cs
               2 File(s)                208 bytes
               2 Dir(s)  341 216 243 712 bytes free

C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld>
```

12. Wpisz poniższe polecenie, aby zbudować i uruchomić aplikację:

```
dotnet run
```

Zobaczysz całą serię komunikatów, gdy narzędzia CLI .NET będą pobierać i instalować jedną lub dwie dodatkowe biblioteki oraz certyfikat deweloperski. Na samym końcu powinieneś zobaczyć tekst **Hello World!** Jest to wyjście (wynik działania) twojego programu.

```
Wiersz polecenia
C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld>dotnet run

.NET 6.0 - Zapraszamy!
-----
Wersja zestawu SDK: 6.0.302

Telemetry
-----
Narzędzia platformy .NET zbierają dane użycia, aby ułatwić nam ulepszanie środowiska. Są one zbierane przez firmę Microsoft i udostępniane społeczności. Możesz zrezygnować z telemetry, ustawiając zmienną środowiskową DOTNET_CLI_TELEMETRY_OPTOUT na wartość "1" lub "true" przy użyciu ulubionej powłoki.

Dowiedz się więcej na temat telemetry narzędzi interfejsu wiersza polecenia programu .NET Core : https://aka.ms/dotnet-cli-telemetry

-----
Zainstalowano certyfikat deweloperski HTTPS programu ASP.NET Core.
Aby zaufać certyfikatowi, uruchom polecenie "dotnet dev-certs https --trust" (tylko systemy Windows i MacOS).
Dowiedz się więcej o protokole HTTPS: https://aka.ms/dotnet-https

-----
Napisz swoją pierwszą aplikację: https://aka.ms/dotnet-hello-world
Dowiedz się, co nowego: https://aka.ms/dotnet-whats-new
Zapoznaj się z dokumentacją: https://aka.ms/dotnet-docs
Raportuj problemy i znajdź źródło w serwisie GitHub: https://github.com/dotnet/core
Użyj polecenia "dotnet --help", aby wyświetlić dostępne polecenia, lub odwiedź stronę https://aka.ms/dotnet-cli

-----
Hello World!

C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld>
```

13. Ponownie wywołaj polecenie **dotnet run**. Tym razem narzędzia .NET CLI nie muszą już pobierać żadnych bibliotek ani certyfikatów, gdyż zostały już one zainstalowane. Zamiast tego powinieneś zobaczyć jedynie komunikat **Hello World!**

Aplikacja Hello World! jest krótka i łatwo ją było napisać, ale konieczne było dodanie pliku projektu, aby narzędzia CLI .NET mogły zrozumieć, jak zbudować i uruchomić program. Takie podejście wystarcza przy prostych aplikacjach, ale bardziej złożone systemy mogą wymagać znaczącej konfiguracji, a tym samym odpowiednio skomplikowanych plików projektu. Dla przykładu aplikacja może wymagać wielu różnych bibliotek i plik projektu musi specyfikować nazwy tych bibliotek, a często również miejsce, z którego narzędzia .NET powinny pobrać te biblioteki. Na szczęście narzędzia .NET potrafią zautomatyzować większość tej złożoności, a przynajmniej ułatwić jej zarządzanie.

W kolejnym ćwiczeniu utworzymy inną wersję Hello World!, ale tym razem sprawimy, aby narzędzia CLI .NET wygenerowały dla nas plik projektu.

Używanie narzędzi CLI .NET do zbudowania i uruchomienia projektu C#

1. W oknie wiersza poleceń powrót do katalogu **Microsoft Press\VCSBS\Chapter 1\HelloWorld** w folderze **Documents**.

```
cd C:\Users\YourName\Documents\Microsoft Press\VCSBS\Chapter 1
```

2. Utwórz nowy katalog o nazwie **HelloWorld2**.

```
mkdir HelloWorld2
```

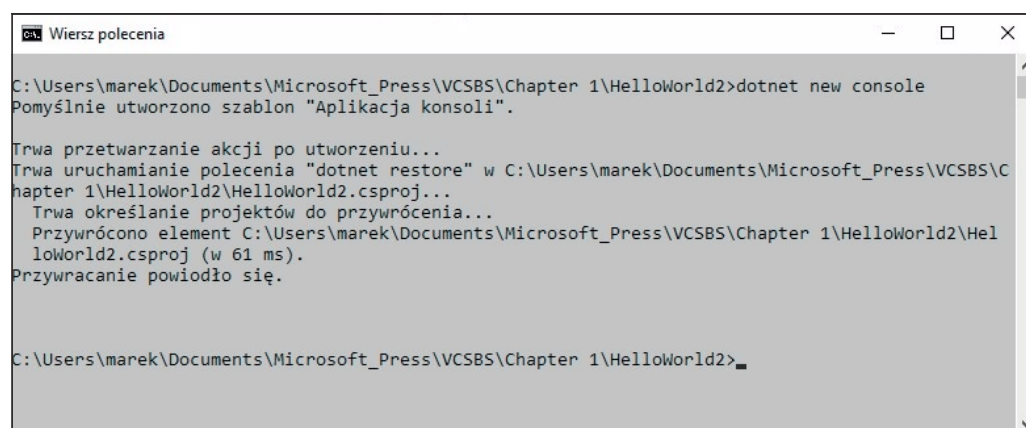
3. Przejdź do katalogu **HelloWorld2**.

```
cd HelloWorld2
```

4. Uruchom poniższe polecenie, aby utworzyć nową aplikację konsolową. Słowo **console** jest nazwą szablonu, którego narzędzia .NET użyją do wygenerowania aplikacji.

```
dotnet new console
```

Zobaczysz kilka komunikatów, gdy narzędzia CLI .NET będą tworzyć plik projektu C# dla nowej aplikacji konsolowej.



```
Wiersz polecenia
C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld2>dotnet new console
Pomyślnie utworzono szablon "Aplikacja konsoli".

Trwa przetwarzanie akcji po utworzeniu...
Trwa uruchamianie polecenia "dotnet restore" w C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld2\HelloWorld2.csproj...
Trwa określanie projektów do przywrócenia...
Przywrócono element C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld2\HelloWorld2.csproj (w 61 ms).
Przywracanie powiodło się.

C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\HelloWorld2>
```

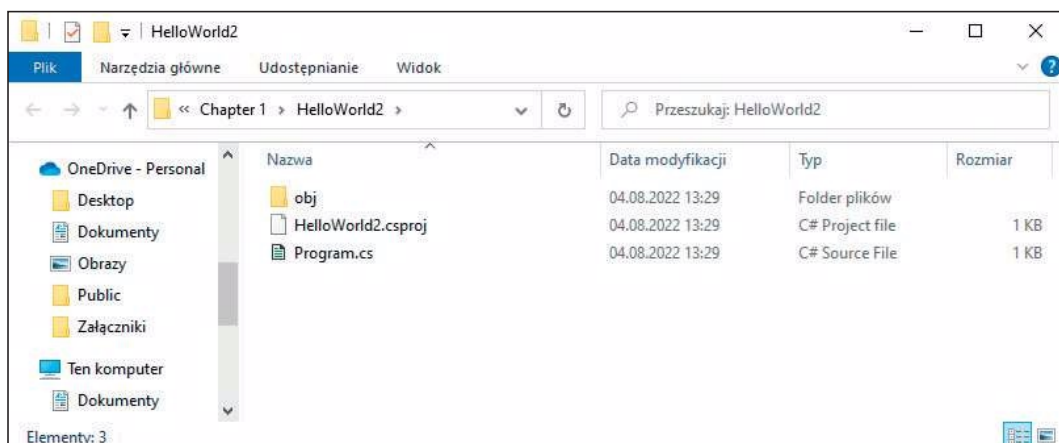


UWAGA Narzędzia CLI .NET udostępniają wiele innych szablonów dla tworzenia aplikacji innych typów, takich jak aplikacje web ASP.NET lub graficzne aplikacje Windows Forms.

5. Napisz **dir**, aby wyświetlić zawartość katalogu HelloWorld2.

Zobaczysz plik projektu o nazwie *HelloWorld2.csproj*, plik kodu C# o nazwie *Program.cs* oraz podrzędny katalog nazwany *obj*. Folder *obj* zawiera pliki konfiguracyjne oraz inne części danych, których narzędzia .NET będą używać do skompilowania i uruchomienia aplikacji.

6. Uruchom Notatnik. Następnie otwórz menu **Plik**, wybierz **Otwórz**, po czym otwórz plik **HelloWorld2.csproj** z folderu **HelloWorld2**. Być może konieczne będzie wybranie opcji **Wszystkie pliki** w liście rozwijanej typów plików, aby zobaczyć pliki znajdujące się w tym folderze.



7. Zawartość pliku powinna wyglądać jak poniżej:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>
</Project>
```

Jak widać, ten plik projektu jest identyczny, jak ten, który utworzyłeś ręcznie w poprzednim ćwiczeniu.

8. Otwórz plik *Program.cs*. Powinien on zawierać poniższy kod:

```
using System;
namespace HelloWorld2
{
    class Program
    {
        static void Main(string[] args)
```



```
        {  
            Console.WriteLine("Hello World!");  
        }  
    }  
}
```

Ten plik programu jest nieco bardziej skomplikowany, niż ten utworzony wcześniej, choć realizuje dokładnie to samo. Plik `Program.cs` definiuje klasę (`class`) o nazwie `Program`, zawierającą metodę `Main`. W języku C# cały kod wykonywalny musi zostać zdefiniowany wewnątrz metody, zaś wszystkie metody muszą należeć do klasy lub struktury (`struct`). Więcej informacji na temat klas poznamy w rozdziale 7, „Tworzenie i zarządzanie klasami oraz obiektami”, a struktury będziemy poznawać w rozdziale 9, „Tworzenie typów wartościowych przy użyciu wyliczeń oraz struktur”.

Metoda `Main` wskazuje punkt wejściowy programu (miejsce, od którego rozpoczyna się wykonywanie programu). Metoda ta powinna zostać zdefiniowana w stylu widocznym w klasie `Program`, jako metoda statyczna (`static`); w przeciwnym razie .NET Framework może jej nie rozpoznać jako punktu startowego aplikacji przy jej uruchamianiu. (Metodom przyjrzymy się szczegółowo w rozdziale 3, „Tworzenie metod i stosowanie zakresów zmiennych”. Rozdział 7 zawiera więcej informacji na temat metod statycznych).

W tym punkcie myślisz pewnie: „Nie utworzyłem ani klasy `Program`, ani metody `Main` w poprzednim przykładzie”. W najnowszej wersji C#, jeśli mamy tak prostą aplikację, jak ta pokazana w pierwszym ćwiczeniu, kompilator C# sam utworzy swój własny punkt wejściowy, jeśli go jawnie nie wskażemy. (Można myśleć o tym jak o syntetycznej metodzie `Main` i jej nazwą rzeczywiście jest `Main`). W wielu przypadkach pozwala to uwolnić się od konieczności pisania niepotrzebnego kodu nagłówkowego (*boilerplate*); wystarczy porównać powyższy kod z programem napisanym w pierwszym ćwiczeniu, aby zrozumieć, co mam na myśli. Jednak szablon aplikacji konsolowej dostarczonej wraz z narzędziami CLI .NET zawsze tworzy klasę `Program` oraz metodę `Main`. Dodatkowo inne typy aplikacji używają innego sposobu oznakowywania punktu wejściowego, który nie wykorzystuje metody `Main`. Szablony dla tych typów aplikacji generują odpowiedni kod rozruchowy. Przykład takiego szablonu zobaczymy w dalszej części tego rozdziału w punkcie „Tworzenie aplikacji graficznej”.

Co do instrukcji `using` i definicji `namespace`, w dalszej części rozdziału dowiesz się więcej na temat tych elementów.

9. Nie zmieniaj niczego; zamknij Notatnik i wróć do kona wiersza poleceń.
10. Skompiluj i uruchom aplikację. Jest to to samo polecenie, którego użyłeś w poprzednim ćwiczeniu.

```
dotnet run
```

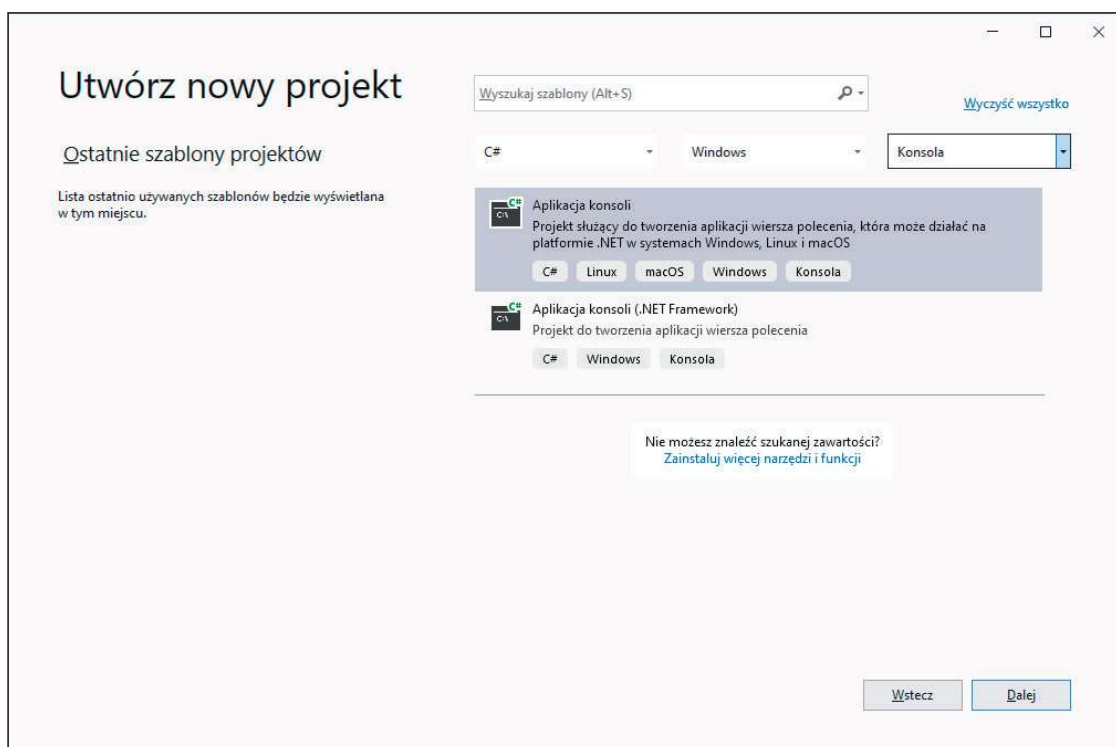
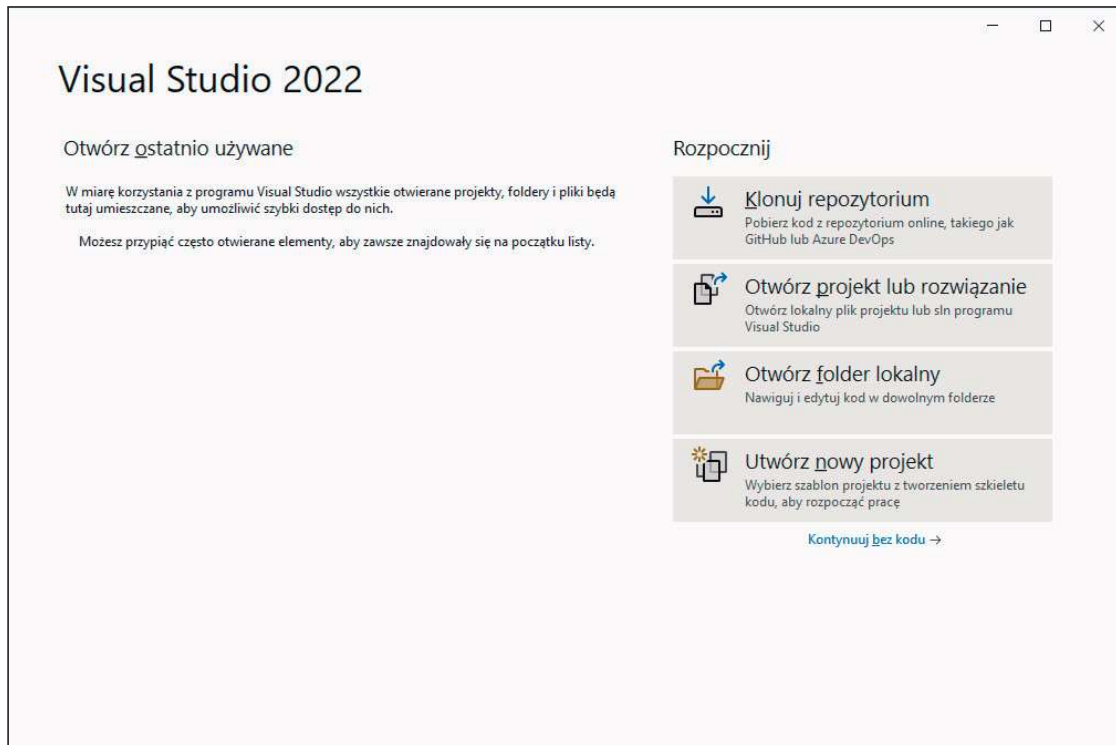
Wynikiem powinien być komunikat `Hello World!` wyświetlony na ekranie.

Zaczynamy programować w środowisku Visual Studio 2022

Teraz, gdy widzieliśmy już podstawową strukturę aplikacji C#, przyszedł czas, aby skierować uwagę na Visual Studio 2022. Visual Studio działający w systemie Windows jest bogato wyposażonym środowiskiem deweloperskim, zawierającym funkcjonalności niezbędne przy tworzeniu zarówno wielkich, jak i drobnych projektów w języku C#. Można nawet konstruować projekty, które gładko łączą moduły napisane w innych językach programowania, takich jak C++, Visual Basic lub F#. W następnym ćwiczeniu uruchomimy środowisko programistyczne Visual Studio 2022 i utworzymy kolejną wersję aplikacji konsolowej Hello World!.

Tworzenie aplikacji konsolowej w Visual Studio 2022

1. W pasku zadań Windows zaznacz **Start**, wpisz **Visual Studio 2022**, po czym naciśnij **Enter**. Alternatywnie możesz kliknąć ikonę **Visual Studio 2022** w menu **Start**.
Uruchomi się program Visual Studio 2022, wyświetlając stronę startową pokazaną na sąsiedniej stronie.
2. W panelu **Rozpocznij** (Get started) wybierz **Utwórz nowy projekt** (Create a new project).
Pojawi się okno dialogowe **Utwórz nowy projekt**. Okno to wylicza szablony, których możemy użyć jako punktu wyjścia dla budowania aplikacji. Szablony te są skategoryzowane według używanego języka programowania i typu aplikacji.
3. Wprowadź poniższe wartości w oknie dialogowym Utwórz nowy projekt, po czym wybierz **Dalej**:
 - a. Na liście rozwijanej **Wszystkie języki** (All languages) wybierz **C#**.
 - b. Na liście rozwijanej **Wszystkie platformy** (All platforms) wybierz **Windows**.
 - c. Na liście rozwijanej **Wszystkie typy projektów** (All project types) wybierz **Konsola**.
 - d. Zaznacz szablon **Aplikacja konsoli (.NET Framework)**.





UWAGA Wersja szablonu *.NET Framework* używa implementacji środowiska .NET specyficznej dla systemu Windows. Wersja *.NET Core* szablonu aplikacji konsolowej używa wersji .NET dostępnej nie tylko w systemie Windows, ale również w systemach Linux i macOS. Tym niemniej na potrzeby tego ćwiczenia wystarczy wersja *.NET Framework* szablonu.

4. W oknie dialogowym **Konfiguruj nowy projekt** wpisz poniższe wartości, po czym wybierz **Utwórz**:
 - a. W polu **Nazwa projektu** (Project name) wpisz **TestHello**.
 - b. W polu **Lokalizacja** wpisz **C:\Users\TwojaNazwa\Documents\Microsoft Press\VCSBS\Chapter 1**.
 - c. Pozostaw zawartość pola **Nazwa rozwiązania** (Solution name) jako **TestHello**.
 - d. Upewnij się, że pole wyboru **Umieść rozwiązanie i projekt w tym samym katalogu** (Place solution and project in the same directory) nie jest zaznaczone.
 - e. Pozostaw domyślną wartość w opcji **Platforma** (Framework).

Konfiguruj nowy projekt

Aplikacja konsoli (.NET Framework) C# Windows Konsola

Nazwa projektu

TestHello

Lokalizacja

C:\Users\marek\Microsoft Press\VCSBS\Chapter 1\

Nazwa rozwiązania ⓘ

TestHello

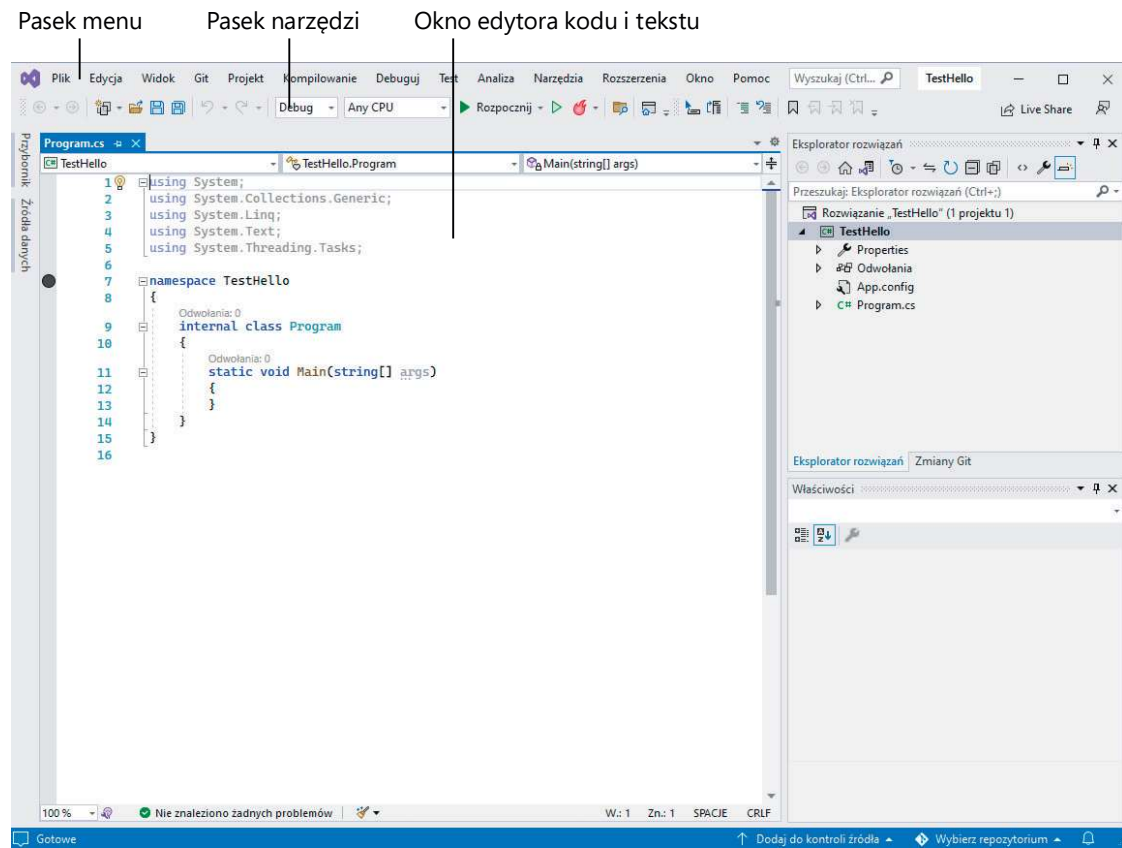
☐ Umieść rozwiązanie i projekt w tym samym katalogu

Platforma

.NET Framework 4.7.2

Wstecz Utwórz

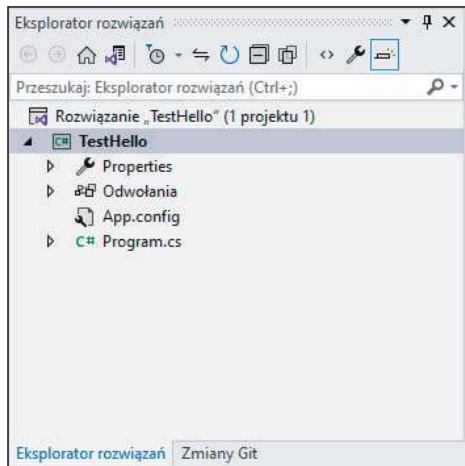
Visual Studio utworzy projekt, wykorzystując szablon aplikacji konsolowej, po czym wyświetli początkowy kod projektu, co pokazuje kolejna ilustracja.



Pasek menu u góry ekranu zapewnia dostęp do funkcjonalności, których będziemy używać w środowisku programistycznym. Dostęp do tych poleceń można uzyskiwać przy użyciu myszy lub skrótów klawiszowych, dokładnie tak samo, jak w każdym innym programie w systemie Windows. Pasek narzędzi jest zlokalizowany poniżej menu. Zapewnia przyciski stanowiące skróty do najczęściej używanych poleceń.

Okno Edytora tekstu i kodu zajmujące większość ekranu wyświetla zawartość plików źródłowych. W wieloplikowym projekcie, gdy edytujemy więcej niż jeden plik, każdy plik źródłowy ma swoją własną zakładkę opisaną nazwą tego pliku. Wybranie zakładki przenosi dany plik źródłowy na wierzch okna edytora.

Panel **Eksplorator rozwiązań** (Solution Explorer) jest widoczny w prawej części IDE, obok okna edytora tekstu i kodu.



Eksplorator rozwiązań oprócz innych elementów wyświetla nazwy plików powiązanych z projektem. Podwójne kliknięcie nazwy pliku w tym panelu powoduje otwarcie tego pliku lub przeniesienie go na wierzch w oknie edytora tekstu i kodu.

5. Zbadajmy wyliczone w Eksploratorze rozwiązań pliki, które Visual Studio 2022 utworzyło jako część naszego projektu:

- **Rozwiązanie 'TestHello'** Jest to plik rozwiązania najwyższego poziomu. Każda aplikacja zawiera pojedynczy plik rozwiązania. Rozwiązanie (*solution*) może zawierać jeden lub więcej projektów, a Visual Studio 2022 tworzy plik rozwiązania jako pomoc w porządkowaniu tych projektów. Jeśli użyjemy Eksploratora plików, aby sprawdzić zawartość folderu Documents\Microsoft Press\VCSBS\Chapter 1\TestHello, zobaczymy, że faktyczna nazwa tego pliku to TestHello.sln.
- **TestHello** Jest to plik projektu C#. Każdy plik projektu odwołuje się do jednego lub więcej plików zawierających kod źródłowy oraz innych artefaktów projektu, takich jak obrazy graficzne. Cały kod źródłowy należący do jednego projektu musi być napisany w tym samym języku programowania. W Eksploratorze plików możemy się przekonać, że ten plik naprawdę nosi nazwę TestHello.csproj i jest umieszczony w folderze \Microsoft Press\VCSBS\Chapter 1\TestHello\TestHello wewnątrz folderu Documents.
- **Properties** Jest to folder wewnątrz projektu TestHello. Jeśli go rozwiniemy (zaznaczając strzałkę na lewo od nazwy Properties), ujrzymy, że zawiera plik o nazwie AssemblyInfo.cs. Jest to plik specjalny, którego możemy użyć w celu dodawania do programu takich atrybutów, jak nazwa autora, data napisania programu i tak dalej. Można również wyspecyfikować dodatkowe atrybuty, aby zmodyfikować sposób uruchamiania programu. Wykorzystywanie tych atrybutów stanowi tematykę zaawansowaną, wykraczającą poza zakres tej książki.
- **References** 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 unikatowa nazwa. W środowisku Microsoft .NET Framework biblioteki te nazywane są *asemblacjami*. Programiści mogą używać asemlacji do umieszczania w nich napisanych przez siebie użytecznych fragmentów kodu w sposób umożliwiający ich dystrybuowanie i używanie przez innych programistów we własnych aplikacjach. Jeśli rozwinimy folder References, to przekonamy się, że zawiera on zestaw domyślnych bibliotek dodanych do projektu przez Visual Studio 2022. Te asemlacje zapewniają dostęp do wielu powszechnie wykorzystywanych funkcji platformy .NET Framework i zostały dostarczone przez firmę Microsoft wraz z Visual Studio 2022. Podczas wykonywania zamieszczonych w tej książce ćwiczeń będziemy mieli okazję zapoznać się dokładniej z wieloma z tych asemlacji.

- **App.config** Jest to plik konfiguracyjny aplikacji. Jest on opcjonalny i nie zawsze musi być obecny. Można w nim wyspecyfikować ustawienia, których aplikacja ma używać w czasie działania, takie jak wersja .NET Framework wykorzystywana do uruchomienia aplikacji. Plik ten będziemy poznawać bliżej w dalszych rozdziałach książki.
- **Program.cs** Jest to plik źródłowy w języku C#, wygenerowany przez szablon i wyświetlony w oknie edytora kodu i tekstu po utworzeniu projektu. Zawartość tego pliku zastąpimy własnym kodem aplikacji.

Pisanie pierwszego programu przy użyciu Visual Studio 2022

Dowiedzieliśmy się wcześniej, że plik Program.cs definiuje klasę o nazwie *Program*, która zawiera metodę o nazwie *Main*.

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



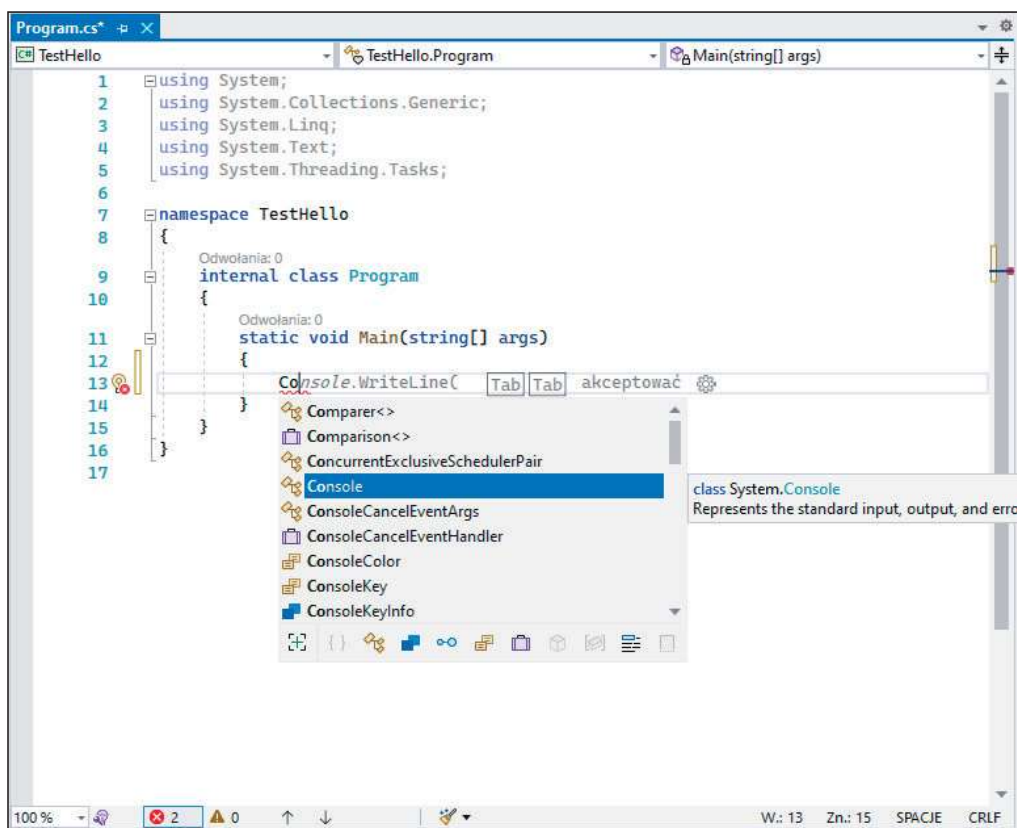
W kolejnych ćwiczeniach zmodyfikujemy kod, aby również wyświetlał w oknie konsoli komunikat „Hello World!” (Witaj świecie!). zbudujemy i uruchomimy naszą aplikację konsolową Hello World przy użyciu Visual Studio. Dowiemy się również, jak wykorzystywać przestrzeń nazw do dzielenia kodu na różne elementy.

Pisanie kodu w Visual Studio przy użyciu funkcji Microsoft IntelliSense

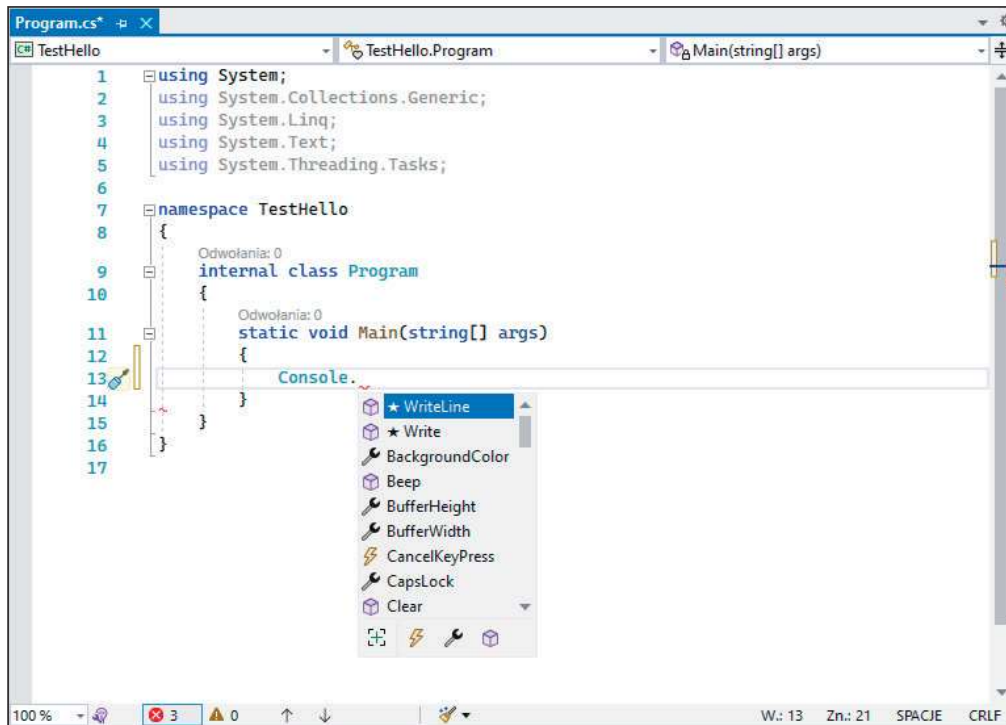
1. W oknie edytora kodu i tekstu, wyświetlającym zawartość pliku Program.cs, umieść kursor wewnątrz metody *Main*, bezpośrednio za otwierającym nawiasem klamrowym {, a następnie naciśnij klawisz Enter, aby utworzyć nową linię.

2. W nowej linii zacznij wpisywać słowo **Console**. Jest to nazwa jeszcze jednej klasy zawartej w jednej z asemblacji dołączonych automatycznie do naszej aplikacji. Klasa ta oferuje metody pozwalające na wyświetlanie komuników w oknie konsoli oraz na odczytywanie danych wprowadzanych z klawiatury.

Po wpisaniu litery **C** wyświetlona zostanie lista podpowiedzi 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.



3. Wpisz znak kropki, bezpośrednio po słowie **Console**.
Spowoduje to wyświetlenie nowej listy podpowiedzi IntelliSense, na której wyświetlane będą nazwy metod, właściwości oraz pól klasy **Console**.



4. Przewiń w dół zawartość tej listy, zaznacz na niej metodę **WriteLine**, a następnie wciśnij klawisz **Enter**. Alternatywnie naciśnij klawisz **Tab**, aby wybrać domyślną metodę (metoda domyślna to ta, która jest używana najczęściej).

Lista podpowiedzi IntelliSense zostanie zamknięta, a do pliku źródłowego zostanie dodane słowo `WriteLine`. Nowa instrukcja powinna teraz wyglądać następująco:

```
Console.WriteLine
```

5. Wpisz znak nawiasu otwierającego `(`. Spowoduje to wyświetlenie kolejnej podpowiedzi podpowiedzi 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 dokładnie w rozdziale 3). Instrukcja powinna teraz wyglądać następująco:

```
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`.



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:

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.

6. Wpisz `)`, a po nim średnik `;`. Instrukcja powinna teraz wyglądać następująco:

```
Console.WriteLine();
```



WSKAZÓWKA Warto wyrobić sobie nawyk, by dopełniające pary znaków – takie jak nawiasy zwykłe `( )` oraz klamrowe `{ }` – wpisywać razem, jeszcze przed wypełnieniem ich treścią. Można łatwo zapomnieć w wpisaniu znaku zamykającego, jeśli odłożymy to do czasu po wprowadzeniu zawartości.

7. Przesuń kursor i wpisz tekst **"Hello *TwojeImię!*"** pomiędzy znakami nawiasów po nazwie metody `WriteLine` (włącznie z cudzysłowami).

Instrukcja powinna teraz wyglądać następująco (oczywiście poza imieniem):

```
Console.WriteLine("Hello John!");
```


Budowanie i uruchamianie aplikacji konsolowej

1. Otwórz menu **Kompilowanie** (Build) i wybierz polecenie **Kompiluj rozwiązanie** (Build Solution).

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

WSKAZÓWKA Jeżeli okno Dane wyjściowe nie jest widoczne, wyświetl je poleceniem **Dane wyjściowe** z menu **Widok**.



W oknie Dane wyjściowe powinien wówczas zostać wyświetlony komunikat podobny pokazanego poniżej, informujący o przebiegu procesu kompilacji programu:

```
Rozpoczęto kompilację...
1>----- Kompilacja rozpoczęta: Projekt: TestHello, Konfiguracja: Debug
Any CPU -----

1> TestHello -> C:\Users\marek\Documents\Microsoft_Press\VCSBS\Chapter 1\
TestHello\TestHello\bin\Debug\TestHello.exe

===== Kompilacja: powiodło się - 1, nie powiodło się - 0,
zaktualizowane - 0, pominięto - 0 =====
```

Jeśli w kodzie źródłowym popełnione zostały jakieś błędy, to zostaną one wymienione w oknie **Lista błędów** (Error List). 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.

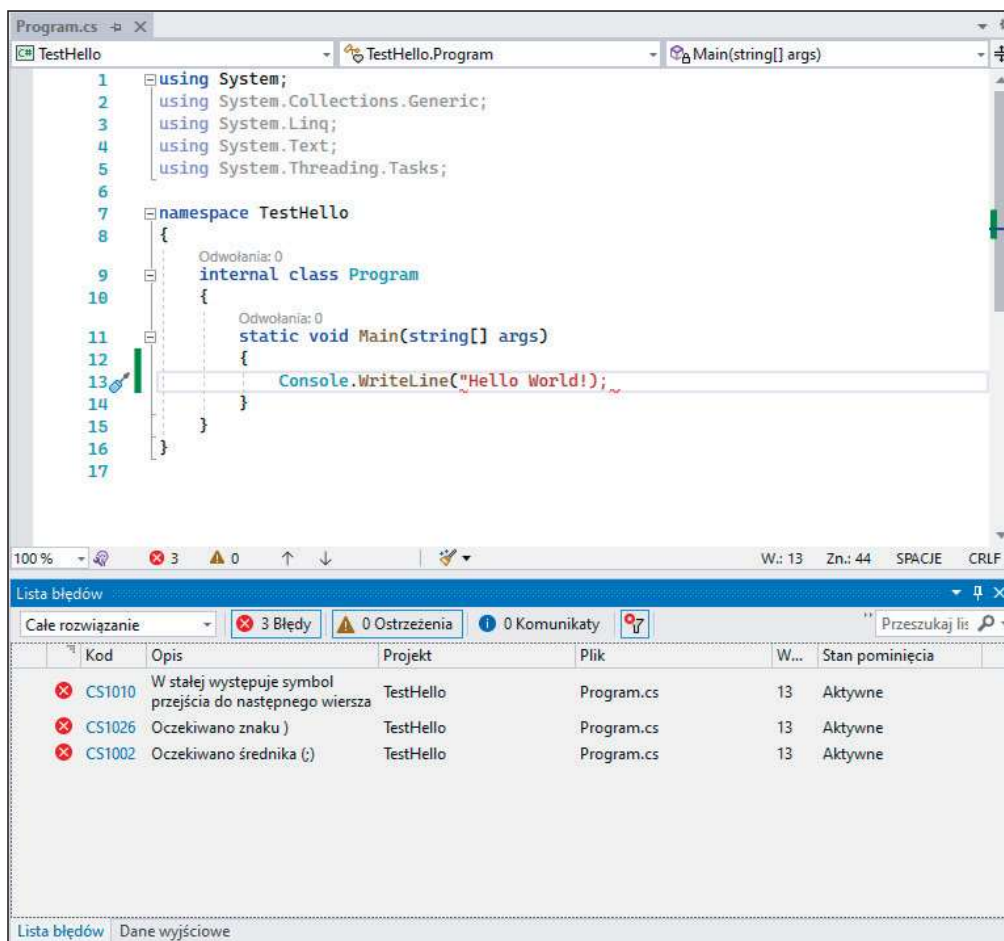
WSKAZÓWKA Zauważ, że w kodzie źródłowym program Visual Studio podkreśla czerwoną falistą linią wszystkie wiersze, które nie będą mogły zostać poprawnie skompilowane. Dwukrotne kliknięcie wybranego elementu w oknie Lista błędów spowoduje umieszczenie kursora w linii, która spowodowała dany błąd.



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.

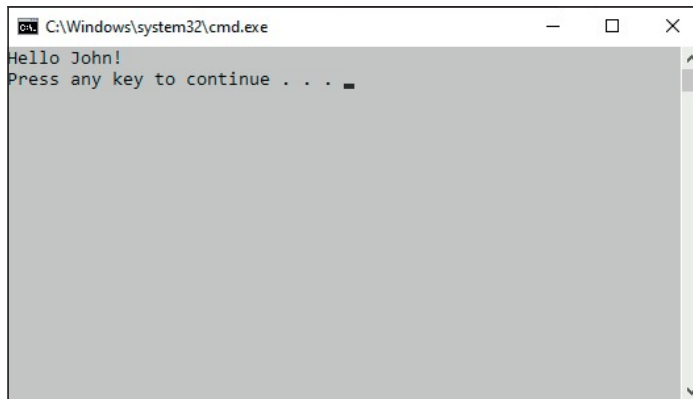


WSKAZÓWKA Nie ma potrzeby jawnego zapisywania pliku źródłowego przed rozpoczęciem procesu budowy/kompilacji, ponieważ polecenie **Kompiluj rozwiązanie** powoduje automatyczne zapisanie pliku źródłowego. Gwiazdka widniejąca obok nazwy pliku na zakładce okna edytora kodu i tekstu oznacza, że dany plik został zmodyfikowany od czasu jego ostatniego zapisania na dysku.



2. W menu **Debuguj** wybierz polecenie **Uruchom bez debugowania** (Start Without Debugging).

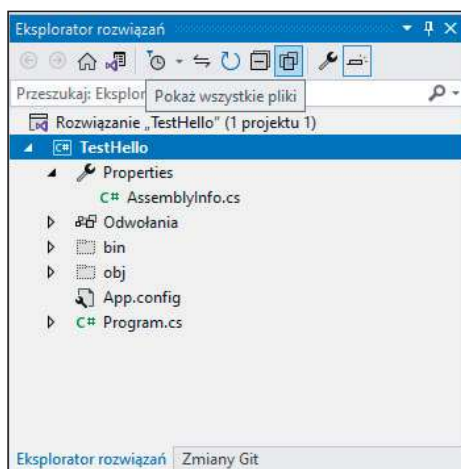
Spowoduje to otwarcie okna wiersza poleceń i uruchomienie programu. Uruchomiony program wyświetli komunikat i będzie oczekiwać na wciśnięcie przez użytkownika dowolnego klawisza, tak jak to zostało pokazane na poniższym rysunku:



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. Gdyby program został uruchomiony przy użyciu polecenia **Rozpocznij debugowanie** (Start Debugging), to aplikacja również zostałaby uruchomiona, ale jej okno wyjściowe zostałoby natychmiast zamknięte, bez oczekiwania na wciśnięcie przez użytkownika dowolnego klawisza.



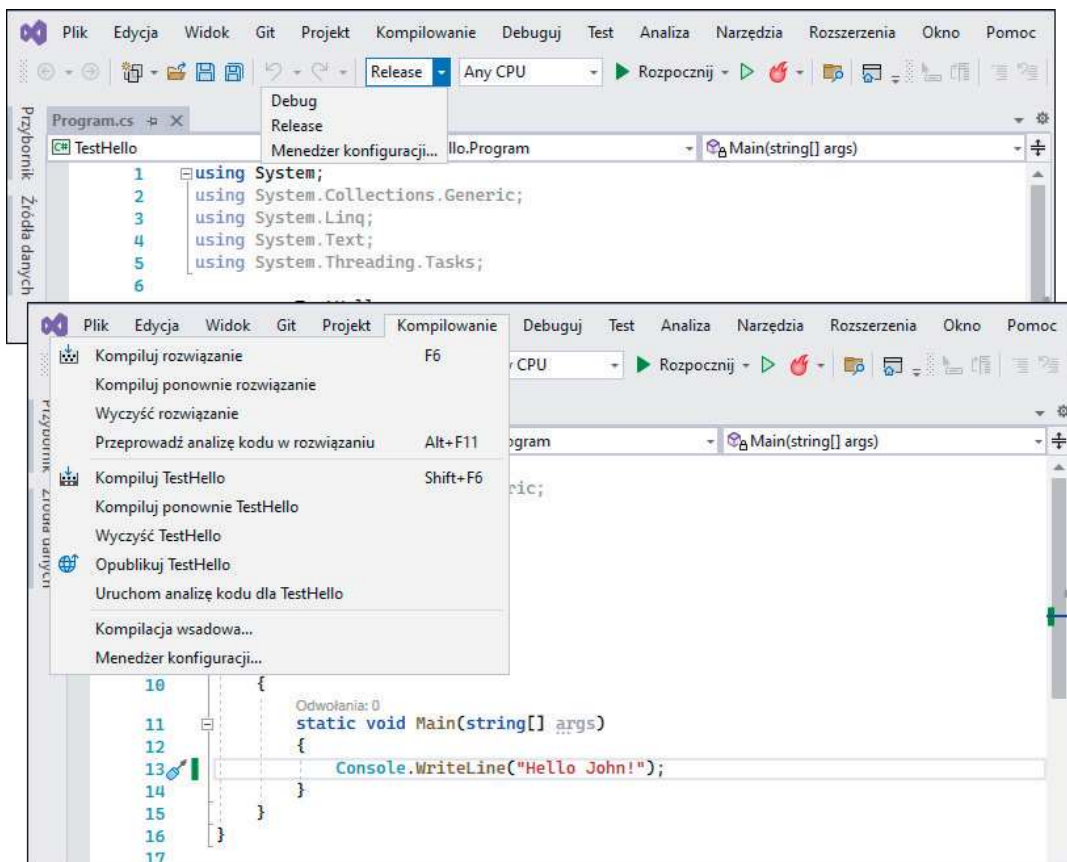
- Upewnij się, że okno wiersza poleceń, w którym wyświetlane są rezultaty działania programu, jest aktywnym oknem, a następnie wciśnij klawisz Enter.
Spowoduje to zamknięcie okna wiersza poleceń i powrót do środowiska programowania Visual Studio 2022.
- W oknie **Eksplorator rozwiązań** kliknij projekt **TestHello** (projekt, a nie rozwiązanie o tej samej nazwie), a następnie kliknij przycisk **Pokaż wszystkie pliki** (Show All Files) w pasku narzędziowym.



Ponad plikiem *Program.cs* zauważymy elementy o nazwach *bin* i *obj*. Wpisy te odpowiadają folderom *bin* i *obj* w folderze projektu (Microsoft Press\Visual CSharp Step By Step\Chapter 1\TestHello\TestHello). Visual Studio utworzyło te foldery podczas budowania aplikacji; zawierają one wykonywalną wersję programu oraz pewne dodatkowe pliki, używane podczas procesu budowania aplikacji oraz podczas jej debugowania.

5. W oknie **Eksplorator rozwiązań** rozwiń gałąź **bin**. Ukaże się wówczas kolejny folder o nazwie *Debug*.
6. Korzystając z Eksploratora 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 Debuguj polecenia **Uruchom bez debugowania**. Pozostałe trzy pliki zawierają informacje, które są używane przez Visual Studio 2022 podczas uruchamiania programu w trybie debugowania – po wybraniu z menu Debuguj polecenia **Rozpocznij debugowanie** (Start Debugging).



Możemy również utworzyć finalną (*release*) kompilację aplikacji. Taka kompilacja nie zawiera informacji debugowania i jest nieco bardziej zwarta (a zapewne

również jest szybsza w działaniu). Gdy z powodzeniem zbudujemy i przetestujemy aplikację, możemy utworzyć finalną wersję i przekazać ją użytkownikom. Aby utworzyć taką wersję, należy wybrać konfigurację **Release** w pasku narzędzi Visual Studio, po czym użyć polecenia **Kompiluj rozwiązanie** z menu **Kompilowanie**.

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:

- W przypadku dużych programów utrzymywanie ich kodu oraz zrozumienie sposobu działania staje się trudniejsze niż w przypadku małych programów.
- 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 kompilowania projektu spowodowanego konfliktem dwóch lub więcej nazw. Przykładem może być próba utworzenia dwóch klas o takiej samej nazwie. Sytuacja komplikuje się jeszcze bardziej, gdy tworzony program odwołuje się do asembacji 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 pomyłone, 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 `NowaPrzestrzenNazw` i asemblacja zawierająca tę klasę zostanie zainstalowana 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 `NowaPrzestrzenNazw.Pozdrowienia`.

Dobre praktyki programowania wymagają, aby wszystkie tworzone klasy były definiowane przy użyciu przestrzeni nazw, do której należą i środowisko Visual Studio 2022 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`. W istocie w pierwszej napisanej przez nas aplikacji użyliśmy w pełni kwalifikowanej nazwy, aby odwołać się do klasy `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 unikatowych nazw, takich jak np. `SystemConsole`. Na szczęście problem ten można rozwiązać stosując w swoich programach dyrektywę `using`. Jeśli cofniemy się do projektu `TestHello` otwartego w Visual Studio 2022 i przyjrzymy się widocznej w oknie edytora kodu i tekstu zawartości pliku `Program.cs`, to zauważymy na początku tego pliku następującą linię:

```
using System;
```

Ten wiersz to dyrektywa `using`. Dyrektywa ta 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 przestrzenią nazw `System`. Przestrzeń nazw `System` jest używana na tyle często, że Visual Studio 2022 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`.



UWAGA Można zauważyć, że niektóre z dyrektyw `using` jest wyszarzonych. Dyrektywy te odpowiadają przestrzeniom nazw, których aplikacja aktualnie nie używa. Jeśli nie będziemy ich potrzebować również po zakończeniu pisania kodu, można je będzie bezpiecznie usunąć. Jednak jeśli będziemy później potrzebować elementów zawartych w tych przestrzeniach nazw, konieczne będzie ich ponowne dodanie do projektu.

Przestrzenie nazw i asemblacje

Dyrektywa `using` powoduje po prostu włączenie elementów ze wskazanej przestrzeni nazw do bieżącego zasięgu, uwalniając programistę od konieczności używania w swoim kodzie w pełni kwalifikowanych nazw klas. Klasy są kompilowane w *aseemblacje* (ang. *assemblies*). Binarna asemblacja to plik, który zwykle ma rozszerzenie `.dll`, choć ściśle rzecz biorąc, również programy wykonywalne z rozszerzeniem `.exe` są asemblacjami.

Asemblacja może zawierać wiele klas. Klasy składające się na bibliotekę klas platformy .NET Framework, takie jak np. `System.Console`, są dostarczane w asemblacjach instalowanych na komputerze razem z Visual Studio. Jak wkrótce się przekonamy, biblioteka klas .NET Framework zawiera tysiące różnych klas. Gdyby wszystkie te klasy znajdowały się w jednym zestawie binarnym, miałby on olbrzymie rozmiary i był trudny do utrzymania. (Gdyby firma Microsoft uaktualniła tylko jedną metodę pojedynczej klasy, musiałaby na nowo rozesłać całą bibliotekę klas do wszystkich programistów!)

Z tego względu biblioteka klas platformy .NET Framework została podzielona na kilka mniejszych asemblacji, odpowiadających różnym obszarom funkcjonalnym, z którymi związane są zawarte w nich klasy. Istnieje więc „zasadnicza” asemblacja (zestaw ten nosi nazwę *mscorlib.dll*), który zawiera wszystkie powszechnie używane klasy, takie jak `System.Console`, a także inne asemblacje, zawierające klasy służące do manipulowania bazami danych, korzystania z usług webowych, tworzenia graficznego interfejsu użytkownika itd.

UWAGA Platforma .NET Framework (będąca czymś innym, niż .NET Core) zawiera swoją własną „rdzeniową” asemblację o nazwie *mscorlib.dll*, która zawiera wiele klas i metod, które mogą się wydawać podobne do tych zawartych w *System.Runtime.dll*. Jednak te dwie asemblacje się różnią i nie są wzajemnie wymienialne. Klasy i metody zawarte w *mscorlib* są implementacjami specyficznymi dla systemu Windows, podczas gdy te znajdujące się w *System.Runtime.dll* są międzyplatformowe – zostały zaprojektowane z myślą o działaniu w środowiskach Windows, Linux i macOS. Asemblacja *System.Runtime.dll* obecnie nie zawiera jeszcze w wszystkich funkcjonalności dostępnych *mscorlib*. Nie należy jednak próbować dodawać asemblacji *mscorlib* do aplikacji .NET Core. Jeśli wymagane są funkcjonalności specyficzne dla Windows, należy użyć szablonu opartego na .NET Framework, a nie na .NET Core. Jeśli to wszystko brzmi nazbyt technicznie, nie trzeba się przejmować. Microsoft intensywnie pracuje nad zunifikowaniem obu zbiorów bibliotek. Plan jest taki, aby w przyszłości środowiska .NET Core i .NET Framework zostały zastąpione pojedynczym środowiskiem wykonawczym i zbiorem asemblacji, nazywanym po prostu *.NET*.

Jeśli zamierzamy skorzystać z klasy zawartej w asemblacji, konieczne jest dodanie w projekcie odwołania do niej. Następnie można dodać w kodzie źródłowym

dyrektywę `using`, która spowoduje włączenie do zasięgu elementów z przestrzeni nazw zawartych w danej asemblacji, aby stały się dostępne w naszym kodzie. Gdy do tworzenia aplikacji używamy Visual Studio, wybrany szablon automatycznie zawiera odwołania do odpowiednich asemblacji. Aby dodać odwołania do dodatkowych asemblacji w projekcie .NET Core, użyjemy menedżera pakietów NuGet. Zobaczymy to w działaniu w późniejszych rozdziałach.

Należy w tym miejscu podkreślić, że relacja pomiędzy binarną asemblacją a przestrzenią nazw niekoniecznie musi być relacją typu 1:1. Pojedyncza asemblacja może zawierać klasy zdefiniowane w wielu różnych przestrzeniach nazw, a pojedyncza przestrzeń nazw może rozciągać się na kilka asemblacji. Wszystko to może początkowo wydawać się bardzo zagmatwane, ale szybko można się do tego przyzwyczaić.

Komentowanie kodu

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 ukośnika i gwiazdki `/*`. Po napotkaniu tych znaków kompilator pomija wszystko, aż do napotkania sekwencji gwiazdki i ukośnika `*/`, która może znajdować się w pliku źródłowym nawet o wiele linii dalej, jak poniżej:

```
/* To jest komentarz wielowierszowy  
Cały tekst zostanie zignorowany przez kompilator  
aż do sekwencji znaków zamykających komentarz */
```

Zdecydowanie zachęcam do dokumentowania własnego kodu źródłowego przy użyciu niezbędnej liczby wyczerpujących komentarzy. Dodatkowo można używać komentarzy do chwilowego wyłączania („wykomentowywania”) jednej lub kilku linii kodu, co typowo jest stosowane podczas testowania i debugowania.

Tworzenie aplikacji graficznej

Dotychczas użyliśmy Visual Studio 2022 do utworzenia i uruchomienia prostej aplikacji konsolowej. Środowisko programowania Visual Studio 2022 zawiera także wszystko, czego będziemy potrzebować do tworzenia graficznych aplikacji dla systemów Windows 10 i Windows 11. Szablony te noszą nazwę aplikacji Universal Windows Platform (UWP), ponieważ umożliwiają tworzenie programów, które można uruchamiać na dowolnym urządzeniu systemu Windows, takim jak komputer, tablet czy telefon. Graficzny interfejs użytkownika aplikacji Windows można projektować interaktywnie. Oprogramowanie Visual Studio 2022 wygeneruje następnie odpowiednie instrukcje programu, implementujące zaprojektowany interfejs użytkownika.

Visual Studio 2022 oferuje dwa rodzaje widoków dla aplikacji graficznych: 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.

Następny zestaw ćwiczeń demonstruje tworzenie aplikacji graficznej przy użyciu Visual Studio 2022. 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.

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

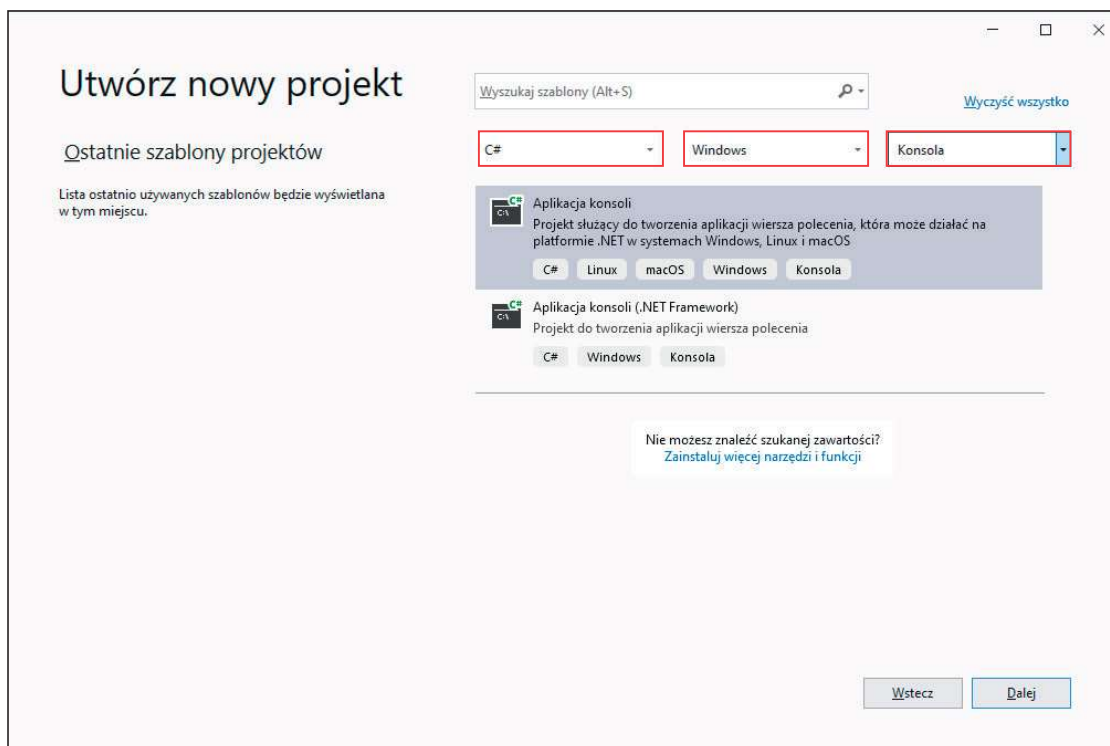


Tworzenie aplikacji graficznej w środowisku Visual Studio 2022

1. O ile Visual Studio 2022 nadal jest uruchomione po poprzednim ćwiczeniu, otwórz menu **Plik** i i wybierz **Nowy, Projekt**. Jeśli zamknąłeś Visual Studio, uruchom je i wybierz **Utwórz nowy projekt**.
2. W oknie dialogowym Utwórz nowy projekt wybierz **C#** na liście rozwijanej języków, **Windows** na liście platform oraz **UWP** na liście typów projektów. Następnie wybierz szablon

WSKAZÓWKA Jeśli nie pojawi się żaden szablon, należy kliknąć łącze **Zainstaluj więcej funkcji** w oknie dialogowym, po czym wybrać segment **Opracowywanie zawartości dla platformy uniwersalnej**.





3. W oknie dialogowym **Konfiguruj nowy projekt** wprowadź poniższe wartości, po czym kliknij **Utwórz**:
 - a. Ustaw nazwę projektu jako **HelloUWP**.
 - b. W polu **Lokalizacja** wpisz **C:\Users\TwojaNazwa\Documents\Microsoft Press\VCSBS\Chapter 1**.
 - c. Z listy rozwijanej **Rozwiązanie** wybierz **Utwórz nowe rozwiązanie**.



UWAGA Lista rozwijana Rozwiązanie pojawia się tylko wtedy, gdy wcześniej edytowa-
liśmy inną aplikację w Visual Studio 2022. Nie pojawia się, jeśli właśnie uruchomiliśmy
Visual Studio.

- d. W polu **Nazwa rozwiązania** pozostaw **HelloUWP**.
- e. Upewnij się, że pole wyboru **Umieść rozwiązanie i projekt w tym samym katalogu**
nie jest zaznaczone.

W tym momencie pojawi się okno dialogowe z pytaniem, w której kompilacji (build) Windows 10 ma być uruchamiana tworzona aplikacja. Późniejsze kompila-
cje Windows 10 udostępniają więcej i nowszych funkcjonalności. Firma Microsoft
zaleca wybieranie najnowszej wersji Windows 10 jako docelowej, ale jeśli pro-
jektujemy aplikację wewnętrzną przedsiębiorstwa, która musi również działać
w starszych wersjach, powinniśmy wybrać najstarszą aktualnie używaną wersję