

O'REILLY®



C# 6.0

Leksykon
kieszonkowy

Helion 

Joseph Albahari,
Ben Albahari

Tytuł oryginału: C# 6.0 Pocket Reference: Instant Help for C# 6.0 Programmers

Tłumaczenie: Przemysław Szeremiota

ISBN: 978-83-283-2446-6

© 2016 Helion S.A.

Authorized Polish translation of the English edition C# 6.0 Pocket Reference ISBN 9781491927410 © 2015 Joseph Albahari, Ben Albahari.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/ch6lek>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Konwencje typograficzne	5
Korzystanie z przykładowych programów	6
Pierwszy program w C#	7
Składnia	10
System typów	13
Typy liczbowe	21
Typ wartości logicznych i operatory logiczne	28
Znaki i ciągi znaków	30
Tablice	34
Zmienne i parametry	38
Operatory i wyrażenia	45
Operatory na typach z dopuszczalną wartością pustą	50
Instrukcje	51
Przestrzenie nazw	58
Klasy	62
Dziedziczenie	74
Typ object	81
Struktury	86
Modyfikatory dostępu	86
Interfejsy	88
Typy wyliczeniowe	91
Typy zagnieżdżone	94
Uogólnienia	94
Delegaty	102
Zdarzenia	108
Wyrażenia lambda	113
Metody anonimowe	116
Wyjątki i instrukcja try	117

Enumeratory i iteratory	125
Typy z dopuszczalną wartością pustą	130
Przeciążanie operatorów	134
Metody rozszerzające	137
Typy anonimowe	139
LINQ	139
Wiązanie dynamiczne	161
Atrybuty	169
Atrybuty wywołania	172
Funkcje asynchroniczne	174
Wskaźniki i kod nienadzorowany	182
Dyrektywy preprocesora	186
Dokumentacja XML	188
Skorowidz	191
O autorach	200

Funkcje asynchroniczne

Zaczynając od wersji 5.0 języka C# na bazie słów kluczowych `await` i `async` wprowadzono mechanizmy asynchronicznego wywołania funkcji; *programowanie asynchroniczne* polega na tym, że funkcje o długim czasie wykonania jak najszybciej zwracają sterowanie do miejsca wywołania, a swoją właściwą pracę wykonują w tle. Dla porównania wywołanie *synchroniczne* oznacza, że powrót z wywołania funkcji następuje dopiero po wykonaniu całości operacji realizowanych przez funkcję. Programowanie asynchroniczne promuje *współbieżność*: długotrwałe zadania są wykonywane *współbieżnie* z podstawowym przebiegiem wykonania programu wywołującego. Implementacja wywołań asynchronicznych odbywa się albo na bazie osobnego wątku, w którym wykonuje się właściwa część zadań funkcji asynchronicznych (w przypadku zadań wymagających obliczeniowo), albo na bazie mechanizmu wywołań zwrotnych (w przypadku operacji wejścia-wyjścia).

Uwaga

Programowanie wielowątkowe, współbieżne i asynchroniczne to same w sobie obszernie zagadnienia: w książce *C# 6.0 in a Nutshell* poświęcono im całe dwa rozdziały; szerokie omówienie znajduje się też pod adresem <http://albahari.com/threading>.

Weźmy za przykład poniższą metodę synchroniczną, wykonującą długo-trwałe obliczenia:

```
int ComplexCalculation()
{
    double x = 2;
    for (int i = 1; i < 100000000; i++)
        x += Math.Sqrt(x) / i;
    return (int)x;
}
```

Wywołanie powyższej metody blokuje wywołującego na co najmniej kilka sekund, aż do obliczenia i zwrócenia wyniku:

```
int result = ComplexCalculation();
// Jakiś czas później:
Console.WriteLine(result); // 116
```

Środowisko uruchomieniowe CLR definiuje (w `System.Threading.Tasks`) klasę o nazwie `Task<TResult>`, reprezentującą operację, która kończy się w przyszłości. Instancję `Task<TResult>` dla zadania obliczeniowego można utworzyć za pośrednictwem metody `Task.Run`, która nakazuje środowisku CLR uruchomienie wskazanego delegatu w osobnym wątku, wykonywanym współbieżnie z wątkiem wywołującego:

```
Task<int> ComplexCalculationAsync()
{
    return Task.Run (() => ComplexCalculation());
}
```

Powyższa metoda staje się *asynchroniczna*, ponieważ zwraca sterowanie do wywołującego natychmiast po wywołaniu `Task.Run`. Potrzebny jest tu jednak jakiś sposób poinstruowania środowiska przez wywołującego, co zrobić po zakończeniu współbieżnej operacji i udostępnieniu jej wyniku. Służy do tego metoda `GetAwaiter` klasy `Task<TResult>`, która z asynchronicznym wywołaniem kojarzy tzw. *kontynuację* (ang. *continuation*):

```
Task<int> task = ComplexCalculationAsync();
var awaiter = task.GetAwaiter();
awaiter.OnCompleted (() =>           // Kontynuacja
{
    int result = awaiter.GetResult();
    Console.WriteLine (result);      // 116
});
```

Powyższe oznacza: „po zakończeniu wywołaj podany delegat”. Nasza kontynuacja najpierw pobiera wynik operacji asynchronicznej (`GetResult`; wywołanie to może też przerzucić wyjątek rzucony przez wywołanie asynchroniczne). Pobrany wynik jest następnie wypisywany na wyjście programu za pomocą wywołania `Console.WriteLine`.

Słowa kluczowe `await` i `async`

Słowo `await` upraszcza kojarzenie kontynuacji z wywołaniami asynchronicznymi. Począwszy od najprostszego scenariusza, kompilator rozwija blok:

```
var wynik = await wyrażenie;
instrukcja/instrukcje;
```

do czegoś w rodzaju:

```
var awaiter = wyrażenie.GetAwaiter();
awaiter.OnCompleted (() =>
{
    var wynik = awaiter.GetResult();
    instrukcja/instrukcje;
});
```

Uwaga

Kompilator dodatkowo emituje kod optymalizujący scenariusz synchronicznego (natychmiastowego) zakończenia operacji; operacja asynchroniczna kończy się od razu najczęściej w wyniku działania wewnętrznego mechanizmu cache, kiedy to wynik operacji jest gotowy w momencie wywołania.

Dzięki temu możemy swoją asynchroniczną metodę `ComplexCalculationAsync` wywołać następująco:

```
int result = await ComplexCalculationAsync();
Console.WriteLine (result);
```

Żeby całość skompilować, trzeba jeszcze dodać modyfikator `async` do deklaracji funkcji zawierającej wywołanie z `await`:

```
async void Test()
{
    int result = await ComplexCalculationAsync();
    Console.WriteLine (result);
}
```

Modyfikator `async` informuje kompilator, aby słowo `await` traktować jako słowo kluczowe, a nie identyfikator widoczny wewnątrz metody (służy to zabezpieczeniu kodu przed kompilacją kompilatorem starszym niż C# 5.0, w którym uznanie słowa `await` za zwyczajny identyfikator spowodowałoby niepoprawną kompilację). Modyfikator `async` można stosować tylko wobec metod (i wyrażeń lambda) bez wartości zwracanych albo zwracających wartości typu `Task` bądź `Task<TResult>`.

Uwaga

Modyfikator `async` działa podobnie jak modyfikator `unsafe` w tym sensie, że nie ma wpływu na sygnaturę czy widoczność metody; wpływa wyłącznie na to, jak metoda jest interpretowana i realizowana *wewnątrz*.

Metody z modyfikatorem `async` są nazywane *funkcjami asynchronicznymi*, bo też zazwyczaj są w istocie asynchroniczne. Aby to uwidocznić, zobaczymy, jak przebiega wykonanie funkcji asynchronicznej.

Po napotkaniu wyrażenia `await` sterowanie (zazwyczaj) wraca do wywołującego (podobnie jak w przypadku instrukcji `yield return` w iteratorach), ale jeszcze zanim to nastąpi, środowisko wykonawcze kojarzy z oczekującym na wykonanie zadaniem kontynuację, aby już po zakończeniu wykonania zadania możliwe było podjęcie wykonania od określonego miejsca programu. Jeśli zadanie asynchroniczne wykona się błędnie, rzucony przez nie wyjątek zostanie przetrzucony do kontynuacji (jak po wywołaniu `GetResult`); jeśli zadanie zwróci wartość, zostanie ona przypisana do wyrażenia `await`.

Uwaga

Wbudowana implementacja metody `OnComplete` kontynuacji oczekującej na zakończenie wykonania asynchronicznego gwarantuje zachowanie bieżącego kontekstu *synchronizacji* (jeśli takowy istnieje). W praktyce oznacza

to, że w aplikacjach z rozbudowanym interfejsem użytkownika (WPF, WinRT, Silverlight czy Windows Forms), przy oczekiwaniu na operację na interfejsie użytkownika, kontynuacja będzie się wykonywać w tym samym wątku co sama operacja; jest to bardzo pomocne w zachowaniu bezpieczeństwa wątkowego aplikacji.

Wyrażenie występujące za `await` to zazwyczaj zadanie, czyli egzemplarz klasy `Task`; może to być jednak dowolny obiekt z metodą `GetAwaiter` zwracającą obiekt implementujący interfejs `INotifyCompletion.OnCompleted` i z metodą `GetResult` o odpowiedniej sygnaturze (oraz z właściwością `IsCompleted`, służącą do sprawdzania możliwości wykonania synchronicznego).

Zauważmy, że wyrażenie `await` jest obliczane jako wartość typu `int`; spowodowane jest to tym, że oczekiwaliśmy na asynchroniczne wykonanie zadania typu `Task<int>` (którego metoda `GetAwaiter().GetResult()` zwraca wartość typu `int`).

Możliwe jest również oczekiwanie na wykonanie zadania nieuogólnionej klasy `Task`; wyrażenie `await` jest wtedy wyrażeniem pustym:

```
await Task.Delay (5000);  
Console.WriteLine ("Minęło pięć sekund!");
```

Metoda `Task.Delay` jest statyczną metodą zwracającą obiekt `Task` kończący wykonanie po odczekaniu określonej liczby milisekund. Synchronicznym odpowiednikiem wywołania `Task.Delay` jest wywołanie `Thread.Sleep`.

Klasa `Task` to nieuogólniona klasa bazowa dla `Task<TResult>` i funkcjonalnie jest równoważna klasie `Task<TResult>` we wszystkim poza wartością zwracaną.

Zachowanie lokalnego kontekstu

Największą siłą wyrażen `await` jest to, że mogą występować w niemal dowolnym miejscu kodu; w szczególności wyrażenie `await` (w obrębie funkcji asynchronicznej) może się pojawiać w miejscu, gdzie dozwolone jest wystąpienie dowolnego wyrażenia za wyjątkiem wyrażenia blokady, kontekstu nienadzorowanego, wnętrza bloku `catch` lub bloku `finally`, kodu nienadzorowanego czy, wreszcie, punktu wejścia do wykonania (czyli głównej metody programu).

W poniższym przykładzie stosujemy wyrażenie `await` w ciele pętli:

```
async void Test()  
{  
    for (int i = 0; i < 10; i++)  
    {  
        int result = await ComplexCalculationAsync();  
    }  
}
```



```
        Console.WriteLine (result);
    }
}
```

Po pierwszym wywołaniu `ComplexCalculationAsync` sterowanie powraca do miejsca wywołania za wyrażeniem `await`. Kiedy metoda zakończy działanie (albo rzuci wyjątek), wykonanie zostanie wznowione w przerwanej miejscy, z zachowaniem wartości zmiennych lokalnych, w tym licznika pętli. Kompilator uzyskuje ten efekt poprzez translację powyższego kodu na maszynę stanową, podobnie jak w przypadku iteratorów.

Nieobecność słowa kluczowego `await` oznaczałaby, że to programista musiałby jawnie oprogramować taką maszynę stanową ze starannym ręcznym stosowaniem kontynuacji; tradycyjnie to właśnie ten element programowania asynchronicznego przysparza najwięcej trudności.

Pisanie funkcji asynchronicznych

W każdej funkcji synchronicznej można zastąpić pusty typ zwracany instancją klasy `Task`, a funkcja ta stanie się w pełni *użyteczną* funkcją asynchroniczną; żadne inne zmiany nie są konieczne:

```
async Task PrintAnswerToLife()
{
    await Task.Delay (5000);
    int answer = 21 * 2;
    Console.WriteLine (answer);
}
```

Zauważmy, że w ciele metody nie ma jawnej instrukcji zwracającej obiekt zadania (`Task`). Zajmuje się tym kompilator, to on również obsługuje sygnalizację zakończenia bądź przerwania wykonania metody (w przypadku wyjątku). Dzięki temu można łatwo tworzyć łańcuchy wywołań asynchronicznych:

```
async Task Go()
{
    await PrintAnswerToLife();
    Console.WriteLine ("Gotowe");
}
```

(A ponieważ metoda `Go` zwraca `Task`, `Go` sama w sobie jest również asynchroniczna). Kompilator rozwinię funkcję asynchroniczną zwracającą zadanie współbieżne `Task` do postaci kodu, który (pośrednio) wykorzystuje klasę `TaskCompletionSource` do utworzenia zadania współbieżnego wraz z sygnalizacją zakończenia bądź błędu.

Uwaga

`TaskCompletionSource` to typ środowiska uruchomieniowego CLR, pozwalający na tworzenie zadań współbieżnych pozostających pod ręczną kontrolą programisty, a także zajmujący się sygnalizacją zakończenia wykonania współbieżnego albo błędu wykonania. W przeciwieństwie do `Task.Run`, `TaskCompletionSource` nie wiąże wątku, w którym występuje wywołanie, na czas wykonania operacji. `TaskCompletionSource` znajduje też zastosowanie przy pisaniu metod obsługujących wejście-wyjście i zwracających zadania współbieżne (jak `Task.Delay`).

Chodzi o zapewnienie, by kiedy metoda asynchroniczna zwracająca zadanie współbieżne zakończy wykonanie, sterowanie mogło zostać przekazane (za pośrednictwem kontynuacji) do miejsca, w którym ktoś na to oczekuje.

Zwracanie `Task<TResult>`

Jeśli ciało funkcji synchronicznej zwraca wartość uogólnionego typu `TResult`, można ją zamienić na asynchroniczną poprzez zamianę wartości zwracanej na `Task<TResult>`:

```
async Task<int> GetAnswerToLife()
{
    await Task.Delay(5000);
    int answer = 21 * 2;
    // answer to int, więc nasza metoda zwraca Task<int>
    return answer;
}
```

Działanie metody `GetAnswerToLife` można zademonstrować poprzez wywołanie jej wewnątrz wcześniej zdefiniowanej metody `PrintAnswerToLife` (która z kolei jest wywoływana we wnętrzu `Go`):

```
async Task Go()
{
    await PrintAnswerToLife();
    Console.WriteLine("Gotowe");
}
async Task PrintAnswerToLife()
{
    int answer = await GetAnswerToLife();
    Console.WriteLine(answer);
}
async Task<int> GetAnswerToLife()
{
    await Task.Delay(5000);
    int answer = 21 * 2;
    return answer;
}
```

Funkcje asynchroniczne upodobniają programowanie asynchroniczne do klasycznego programowania synchronicznego. Poniżej zaprezentowany jest synchroniczny równoważnik powyższego łańcucha wywołań, w którym wywołanie `Go()` daje taki sam efekt po blokującym (synchronicznym) odczekaniu pięciu sekund:

```
void Go()
{
    PrintAnswerToLife();
    Console.WriteLine ("Gotowe");
}
void PrintAnswerToLife()
{
    int answer = GetAnswerToLife();
    Console.WriteLine (answer);
}
int GetAnswerToLife()
{
    Thread.Sleep (5000);
    int answer = 21 * 2;
    return answer;
}
```

Powyższy przykład ilustruje też podstawową zasadę projektowania funkcji asynchronicznych w C#, czyli pisanie metod najpierw jako synchronicznych, a następnie zastępowanie wywołań *synchronicznych* wywołaniami *asynchronicznymi* z oczekiwaniem na ich wykonanie współbieżne.

Współbieżność

Poznaliśmy najprostszy schemat programowania asynchronicznego, w którym tuż po wywołaniu funkcji asynchronicznych oczekujemy na zakończenie odpowiadających im zadań współbieżnych. W efekcie otrzymujemy program z uszeregowanym wykonaniem, z logicznego punktu widzenia równoważny programowi w pełni synchronicznemu.

Natomiast prawdziwie współbieżne wykonanie implementujemy wtedy, kiedy wywołujemy funkcje asynchroniczne bez momentalnego oczekiwania na ich zakończenie. Na przykład poniższy kod dokona współbieżnego, dwukrotnego wykonania metody `PrintAnswerToLife`:

```
var task1 = PrintAnswerToLife();
var task2 = PrintAnswerToLife();
await task1; await task2;
```

Współbieżny przebieg wykonania programu kończy się w miejscu oczekiwania na zakończenie obu współbieżnych zadań (tam też dochodzi do kolekcjonowania wyników albo rzucenia wyjątków). Klasa `Task` udostępnia statyczną metodę o nazwie `WhenAll`, agregującą oczekiwanie i przez to nieco wydajniejszą. Me-

toda `WhenAll` zwraca zadanie współbieżne, które kończy się, kiedy zakończą się wszystkie zadania współbieżne wskazane w wywołaniu:

```
await Task.WhenAll (PrintAnswerToLife(),  
                    PrintAnswerToLife());
```

Metoda `WhenAll` to tak zwany agregator zadań (ang. *task combinator*; klasa `Task` udostępnia również agregator o nazwie `WhenAny`, kończący wykonanie w momencie zakończenia dowolnego z przekazanych zadań). Agregatory zadań współbieżnych zostały szczegółowo omówione w książce *C# 6.0 in a Nutshell*.

A

akcesor, 112
właściwości, *Patrz:*
właściwość akcesor

aplikacja, 9

argument, *Patrz też:*
parametr
dynamiczny, 168
nazwany, 44
przekazywanie, 40,
41, 42

assembly, *Patrz:* zestaw

atribut, 169
CLSCompliant, 170
obiekt docelowy, 170
ObsoleteAttribute,
169
odwołanie, 172
parametr, 170
własny, 171
wywołanie, 172
XmlElementAttribute,
170

attribute target, *Patrz:*
atribut obiekt
docelowy

B

biblioteka, 9, 10

blok
catch, 117, 118, 119
finally, 117, 119, 120
instrukcji, 7, 12, 51
try, 117

broadcaster, *Patrz:* nadawca

C

caller info attribute,
Patrz: atrybut
wywołanie

ciąg
łączenie, 32
porównywanie, 33
przeszukiwanie, 33
znaków, 31, 32

constructor, *Patrz:*
konstruktor

custom binding, *Patrz:*
wiązanie
niestandardowe

D

dane
składowe, 15, 16
styczne, 100
wejściowe, 8, 63
wyjściowe, 8

delegat, 102, 103, 108
Action, 105
do metody
instancji, 104
stycznej, 105
Func, 105
modyfikowalność, 104
typ, 102
kontrawariancja, 107
kowariancja, 106,
107

uogólniony, 105
zgodność, 106
uogólniony, 107
wielokrotny, 103, 104

domknięcie, 114

dyrektywa
#define, 187
#elif, 187
#else, 187
#endif, 187
#endregion, 187
#error, 186, 187
#if, 187
#line, 187
#pragma, 187
#region, 187
#undef, 187
#warning, 186, 187
preprocesora, 186, 187
using, 59
using static, 60

dziedziczenie, 74, 80, 81,
88, 89
po klasie, 74

E

enumerator, 125
escape sequence,
Patrz: znak sterujący
event, *Patrz:* zdarzenie
exception filter,
Patrz: wyjątek filtr

F

finalizator, 8, 72, 86
finalizer, *Patrz:* finalizator
flaga, 92
fluent syntax, *Patrz:*
wyrażenie składnia
kaskadowa
fully qualified type name,
Patrz: typ nazwa
w pełni kwalifikowana
funkcja
asynchroniczna, 174,
178, 179
niedynamiczna, 168
przesłanianie, 77
składowa, 15, 79,
Patrz też: metoda
synchroniczna, 178
wirtualna, 77
wywołanie, 39
asynchroniczne, 174
synchroniczne, 174

G

garbage collector,
Patrz: mechanizm
odśmieciania
generic method, *Patrz:*
metoda uogólniona
generic type, *Patrz:*
uogólnienie

I

identyfikator, 10
interfejs, 163
indekser, 8, 69
implementowanie, 70
wirtualny, 77
indexer, *Patrz:* indekser
inferencja, 23
instancja, 15
składowa, 16
typu
wartościowego, 39

instantiation,
Patrz: konkretyzacja
instrukcja, 7, 51
blok, *Patrz:* blok
instrukcji
break, 57
continue, 57
deklaracji, 52
do-while, 55
fixed, 183
for, 55, 56
foreach, 35, 55, 57,
125
goto, 57, 58
if, 53
if.else, 53
iteracyjna, *Patrz:*
instrukcja pętli
pętli, 55
return, 57, 58, 128
skoku, 57
switch, 54, 57
throw, 57, 122, 123
try, 117, 120
using, 121
warunkowa, 52
while, 55
wyrażeniowa, 52,
Patrz też: wyrażenie
yield, 128
yield break, 128
interfejs, 82, 88, 90
deklaracja, 88
dziedziczenie, 89
IComparable, 136
IDynamicMetaObject
↳Binding, 163
IDynamicMetaObject
↳Provider, 163, 164
IEnumerable, 35, 101,
125, 127, 128
IEnumerator, 101, 127
implementacja
jawna, 89
ponowna, 90
wirtualna, 90

INotifyCompletion.
↳OnCompleted,
177
rozszerzanie, 138
składowa, 168
System.Collections.
↳Generic.IEnume
↳rable, 128
System.Collections.
↳Generic.IEnume
↳rator, 125, 128
System.Collections.
↳IEnumerable,
126, 128
System.Collections.
↳IEnumerator,
125, 128
System.IDisposable,
121
interpolated string,
Patrz: ciąg znaków
interpolowany
iterator, 126, 127, 128
sekwencja złożona,
129

J

jagged array, *Patrz:* tablica
wyszczerbiona

K

klasa, 9, 62, 88
abstrakcyjna, 78
bazowa, 74, 75, 78
Console, 9
definiowanie, 62
dziedziczenie,
Patrz: dziedziczenie
po klasie
Enumerable, 149, 150
nazwa, 10
object, 82, 84

- pochodna, 74, 75, 78, 80, *Patrz też:*
 - podklasa
- pochodne, 78
- statyczna, 71
- string, 34, 69
- System.Array, 35
- System.Console, 71
- System.EventArgs, 110
- System.Exception, 123
- System.Math, 71
- kod nienadzorowany, 183
- kod XML, 188
- kolejka
 - LIFO, *Patrz:* stos
- kolekcja, 126
- komentarz
 - dokumentujący, 188
 - jednowierszowy, 13
 - wielowierszowy, 13
- kompilator, 9, 10, 187
- komunikat, 187
- konkretyzacja, 15
- konstruktor, 8, 64, 80
 - bezparametrowy, 65, 86
 - kolejność, 81
 - niepubliczny, 65
 - obiektu, 16
 - przeciążanie, 64
 - statyczny, 71
 - wywoływany, 65
- kontrawariancja, 100, 101, 107
- konwencja wielbłądzia, 11
- kowariancja, 100, 101, 106, 107
- kwalfikator, 148
 - global::, 62
- kwantyfikikator, 143, 144

L

- lambda expression, *Patrz:*
 - wyrażenie lambda

- language binding,
 - Patrz:* wiązanie językowe
- Language Integrated Query, *Patrz:* LINQ
- LINQ, 139, 140, 150
 - element, 140
 - sekwencja, 140
 - złączenie,
 - Patrz:* złączenie
- lista, 69
- literal, 8, 12
 - ciągu znaków, 32
 - liczb, 22

M

- mechanizm
 - attributów, 169
 - LINQ, *Patrz:* LINQ
 - odśmieciania, 39, 72
- metoda, 7, 15, 63
 - Aggregate, 148
 - All, 148
 - anonimowa, 116
 - Any, 148
 - AsEnumerable, 148
 - AsQueryable, 148
 - Average, 148
 - BinarySearch, 35
 - Cast, 148
 - CompareTo, 33
 - Concat, 147
 - Contains, 148
 - Copy, 35
 - Count, 148
 - CreateInstance, 35
 - częściowa, 73
 - deklaracja, 40
 - Display, 122
 - Dispose, 121
 - Distinct, 146
 - ElementAt, 147
 - ElementAtOrDefault, 147
 - Empty, 148
 - Equals, 84, 85
 - Except, 147
 - Finalize, 72
 - Find, 35
 - FindLastIndex, 35
 - FindIndex, 35
 - First, 147
 - FirstOrDefault, 147
 - GetHashCode, 85
 - GetLength, 36
 - GetResult, 177
 - GetType, 84
 - GetValue, 35
 - GroupBy, 147
 - GroupJoin, 147, 155, 156
 - IndexOf, 35
 - Insert, 34
 - instancji, 104, 139
 - Intersect, 147
 - IsCapitalized, 137
 - Join, 147, 155
 - Last, 147
 - LastIndexOf, 35
 - LastOrDefault, 147
 - LongCount, 148
 - Main, 9
 - Max, 148
 - Min, 148
 - MoveNext, 144
 - nazwa, 10, 63
 - OfType, 148
 - OnComplete, 176
 - OrderBy, 147, 149
 - OrderByDescending, 147
 - PadLeft, 34
 - PadRight, 34
 - parametr,
 - Patrz:* parametr przeciążanie, 64, 81
 - Range, 148
 - ReferenceEquals, 85
 - Remove, 34
 - Repeat, 148
 - Reverse, 147

- metoda
- rozszerzająca, 137, 139, 141, 168
 - wywołanie
 - kaskadowe, 138
 - Select, 141, 146, 149
 - SelectMany, 146
 - SequenceEqual, 148
 - SetValue, 35
 - Single, 147
 - Skip, 146
 - SkipWhile, 146
 - skrótowa do
 - wyrażenia, 63
 - Sort, 35
 - statyczna, 65
 - Substring, 34
 - Sum, 148
 - sygnatura, 63
 - Take, 146
 - TakeWhile, 146
 - ThenBy, 147
 - ThenByDescending, 147
 - ToArray, 148
 - ToDictionary, 148
 - ToList, 148
 - ToLookup, 148
 - ToLower, 34
 - ToString, 85
 - ToUpper, 34
 - Trim, 34
 - TrimEnd, 34
 - TrimStart, 34
 - Union, 147
 - uogólniona, 96
 - wartość zwracana, 8
 - Where, 146, 149
 - wirtualna, 77, 78
 - WriteLine, 9
 - Zip, 147
- modyfikator
- async, 176, 181
 - await, 176, 177
 - dostępu
 - internal, 87
 - private, 87
 - protected, 87
 - protected internal, 87
 - public, 86
 - out, 40, 42
 - params, 42
 - readonly, 63
 - ref, 40, 41, 169
 - this, 137
 - unsafe, 176
 - virtual, 169
- multimetoda, 168
- multiple dispatch, *Patrz:* multimetoda
- ## N
- nadawca, 108
- namespace, *Patrz:* przestrzeń nazw
- nested type, *Patrz:* typ zagnieżdżony
- null coalescing operator, *Patrz:* operator ??
- nullable type, *Patrz:* typ z dopuszczalną wartością pustą
- null-conditional operator, *Patrz:* operator ?.
- ## O
- obiekt, 38
- inicjalizator, 65
 - instancja, *Patrz:* instancja
 - konstruktor, *Patrz:* konstruktor obiektu
 - nadzorowany, 183
 - przeliczalny, 125
 - System.Object, 81
 - System.Type, 84
 - tablicy, 35
 - tworzenie, 38
 - typ, *Patrz:* typ
 - Type, 97
- obliczenie nadzorowane, 25
- odbiornik komunikatów, 167
- odpakowywanie, 82
- operand, 45
- operator, 8, 12, 45, 47, *Patrz też:* znak
- !=, 29, 49, 93, 132, 134
 - przeciążanie, 85
 - &, 26, 49, 93, 133, 134, 182
 - &&, 29
 - ?., 50
 - ?:, 49
 - ??, 50, 134
 - ^, 26, 49, 93, 133, 134
 - |, 26, 49, 93, 133, 134
 - ||, 29, 49
 - ~, 26, 93, 133
 - +, 32
 - <<, 26
 - ==, 131, 134
 - przeciążanie, 85
 - >>, 26
 - addytywny, 48
 - agregacji, 143, 144, 145
 - All, 143
 - alternatywy dla null, 47
 - Any, 143
 - arytmetyczny, 24, 93
 - as, 49, 76
 - Average, 143
 - await, 48
 - bitowej sumy
 - wyłączającej, 49
 - bitowy, 26, 93
 - Cast, 160
 - checked, 25, 26, 48
 - Concat, 143
 - Contains, 143
 - Count, 143
 - default, 48

dekrementacji, 25
dostępu przez
 wskaźnik, 182, 184
dwuargumentowy, 45
 zapis wrostkowy, 45
elementowy, 142, 144,
 147
Elvis, *Patrz:* operator ?.
Except, 144
First, 142
FirstOrDefault, 142
funkcja, 135
główny, 47
GroupBy, 159, 160
GroupJoin, 155
iloczynu bitowego, 49
iloczynu logicznego,
 49
inkrementacji, 25
Intersect, 144
is, 49, 77
jednoargumentowy,
 45, 48
Join, 155
konwersji, 144, 145
lambda, 47, 49
Last, 142
łączność, 46
 lewostronna, 47
 prawostronna, 47
Max, 143
Min, 143
mnożenia, 12
multiplikatywny, 48
nameof, 47, 73
new, 16, 47
obliczenia
 nadzorowanego,
 Patrz: operator
 checked
odwołania do
 składowej, 12
OfType, 160
pierwszorzędny, 45
pobrania adresu, 182

porównania, 29, 49,
 84, 93, 132, 135
pożyczanie, 131, 132
priorytet, 46, 47
przeciążanie, 134,
 135, 136
przesunięć bitowych,
 48
przypisania, 13, 46, 47,
 49
relacji, 29, 48, 132, 135
Reverse, 142
równości, 29
SelectMany, 154
SequenceEquals, 143
Single, 142
sizeof, 48, 93
Skip, 142
stackalloc, 47
struktury Nullable,
 131, 132, 133
sumy bitowej, 49
sumy logicznej, 49
Take, 142
ToArray, 144
ToDictionary, 144
ToList, 144, 145
ToLookup, 144
trójargumentowy, 45
trójwartościowy, 47
typeof, 47, 97
typu wyliczeniowego,
 93
unchecked, 26, 48
Union, 143
warunkowego dostępu
 do składowej,
 Patrz: operator ?.
warunkowy, 47, 49
Where, 140
wyluskania, 182
XOR, 49
zapytania, 141, 144,
 145, 146, 147
 kaskadowy, 149
zapytania, 141

zbiorów, 143
złączenia, 155

P

pakowanie, 82, 83
parametr, 8, 38, 40, 63,
 114, *Patrz też:*
 argument
 opcjonalny, 43
 typowy, 95
 deklarowanie, 96
 wartość domyślna,
 98
partially qualified name,
 Patrz: przestrzeń nazw
 nazwa częściowa
 kwalifikacja
pętla, *Patrz:* instrukcja
 pętli
platforma .NET
 Framework, 9
plik
 .cs, 9
 .dll, 9
 .exe, 9
 csc.exe, 10
podklasa, 74
podzapytanie, 145
pole, 38, 39, 63
 deklarowanie, 63
 inicjalizacja, 63
 instancji, 38
 publiczne, 67
 statyczne, 38
polimorfizm, 74
preprocesor, 186, 187
 symbol
 #error, 186
 #warning, 186
primary operator,
 Patrz: operator
 pierwszorzędny
programowanie
 asynchroniczne, 174,
 178, 179

property, *Patrz:*
właściwość
protokół, 102
przesłanie, 79
przezeń nazw, 9, 58, 59,
138
alias, 62
deklaracja, 62
globalna, 59
importowanie, 59, 62
nazwa
częściowa
kwalifikacja, 61
przesłanie, 61
zasięg, 61
System, 9, 14
System.Collections, 35
przyjęcie oznaczone, 39
przyrostek
D, 23
L, 23
liczbowy, 23
U, 23

Q

query expression, *Patrz:*
wyrażenie zapytaniowe

R

rectangular array, *Patrz:*
tablica regularna
referencja, 29, 41
dynamic, 165
object, 165
polimorficzna, 74
pusta, 130
rzutowana
jawnie, 75
niejawnie, 75
w dół, 75, 76
w górę, 75
this, 65, 66

S

sealing, *Patrz:*
zapieczętowanie
implementacji
serializacja, 169
słownik, 35, 69, 85, 126
słowo kluczowe, 11
async, 174, 176, 181
await, 174, 175, 177
base, 79
class, 62
const, 70
default, 40, 98
delegate, 116
dynamic, 161
explicit, 134
fixed, 185
implicit, 134
internal, 59
into, 153
kontekstowe, 12
let, 152
namespace, 59
orderby, 158
partial, 72
private, 59
public, 59
stackalloc, 184
static, 16, 71
this, 64, 66, 79
unsafe, 182
var, 44
virtual, 77
stała, 13, 45, 70
deklarowanie, 70
statement, *Patrz:*
instrukcja
sterta, 38
stos, 38, 82, 184
struktura, 86, 88
Nullable, 130
operatory, 131
strumień wejścia-wyjścia, 9
subclass, *Patrz:* podklasa

subscriber, *Patrz:*
subskrybent

T

tablica, 34, 184
deklaracja, 34
dynamiczna, 35
element, 38
inicjalizacja, 35, 37, 39
kopiowanie, 35
liczb całkowitych, 35
liczba wymiarów, 35
mieszająca, 85
nieposortowana, 35
posortowana, 35
regularna, 36
deklaracja, 36
rozmiar, 35
sortowanie, 35
tworzenie, 35
typ, 36
wielowymiarowa, 36
wyszczerbiona, 36, 37
wyszukiwanie, 35
zagnieżdżona,
Patrz: tablica
wyszczerbiona
typ, 13
anonimowy, 139
bazowy, 168
bool, 14, 21, 28, 40
byte, 21, 22, 26
całkowitoliczbowy, 8,
17
przepełnienie, 25
char, 21, 30, 40
ciągu znaków, 21
częściowy, 72
decimal, 21, 22, 23, 28
delegatu, 102
kontrawariancja, 107
kowariancja, 106,
107
uogólniony, 105
zgodność, 106

dookreślony, *Patrz:*
typ zamknięty
dostępność, 88
double, 21, 22, 23, 28
dynamiczny, 165
konwersja, 166
enum, 91
false, 40
float, 21, 22, 23, 28
int, 8, 14, 15, 21, 22, 35
kontrola, 83
konwersja, 24, 136, 166
jawna, 24
niejawna, 24, 26
liczbowy, 21, 40, 86
logiczny, 14, 21, 28, 40
long, 17, 21, 22, 23
nazwa
kolizja, 62
w pełni
kwalifikowana, 59
niedookreślony,
Patrz: typ otwarty
niejawnie
konwertowany, 23
niemodyfikowalny, 69
obiektyowy, 21
object, 21, 81, 82, 165
ObsoleteAttribute,
169, 170
odwołanie, 59
otwarty, 95, 97
predefiniowany, 13,
14, 15, 21, 40
referencyjny, 21, 29, 31,
36, 38, 41, 63, 130
sbyte, 21, 22, 26
short, 21, 22, 26
string, 14, 21, 31, 33
uint, 21, 22, 23
ulong, 21, 22, 23
uogólniony,
Patrz: uogólnienie
ushort, 21, 22, 26
void, 63, 104
wartościowy, 21, 31,
39, 86, 182

wartość
domyślna,
Patrz: wartość
domyślna
wbudowany, *Patrz:*
typ predefiniowany
własny, 14
wnioskowany, 23
wskaźnikowy, 182
wyliczeniowy, 40, 91,
92
konwersja, 92
z dopuszczalną
wartością pustą, 130
konwersja, 131, 133
pakowanie, 131
zagnieżdżony, 94
zamknięty, 95
znakowy, 21, 40
type argument, *Patrz:*
argument typowy
type parameter, *Patrz:*
parametr typowy
operator, 84
argument, 95

U

uogólnienie, 94, 95, 96
ograniczenia, 98
pochodne, 99

V

verbatim string literal,
Patrz: literał ciągu
znaków dosłowny
void expression, *Patrz:*
wyrażenie puste

W

wartość
domyślna, 40, 43, 63,
92
minus 0, 27

minus
nieskończoność, 27
NaN, 27
null, 36, 40, 104, 130
plus nieskończoność,
27
whitespace, *Patrz:* znak
biała spacja
wiązanie
dynamiczne, 161, 162,
163, 164, 165, 167
językowe, 163, 164, 165
niestandardowe, 163
statyczne, 162
według reguł, 163
ze wskazania,
Patrz: wiązanie
niestandardowe
wielorozprowadzanie,
Patrz: multimetoda
wiersz poleceń, 10
właściwość, 8, 66
akcesor, 67, 68
get, 67, 69
poziom dostępności,
69
set, 67, 68, 69
automatyczna, 68
inicjalizacja, 68
Length, 35
Rank, 35
skrótowa do wyrażenia,
68
this, 70
wirtualna, 77
wyłącznie do
odczytu, 67
zapisu, 67
wskaźnik beztypowy, 185
współbieżność, 174, 179,
180
wyjątek
czasu wykonania, 133,
165
DivideByZero
↳ Exception, 118

- wyjątek
 - filtr, 120
 - IndexOutOfRangeException
 - ↳ Exception, 35
 - InvalidCastException, 83
 - NullReference
 - ↳ Exception, 50
 - OverflowException, 25
 - RuntimeBinder
 - ↳ Exception, 165, 166
 - System.Argument
 - ↳ Exception, 124
 - System.ArgumentNull
 - ↳ Exception, 122, 124
 - System.ArgumentOutOfRangeException
 - ↳ Exception, 124
 - System.InvalidOperationException
 - ↳ Exception, 124
 - System.NotImplementedException
 - ↳ Exception, 124
 - System.NotSupportedException
 - ↳ Exception, 124
 - System.ObjectDisposedException, 124
 - typ, 119
 - OutOfMemory
 - ↳ Exception, 119
 - System.Exception, 119
 - WebException, 120
 - zgłaszanie, 122
 - ponowne, 122, 123
- wyrażenie, 45, 52, 63, 65
 - dynamiczne, 167
 - lambda, 113, 114
 - asynchroniczne, 181
 - zmienna
 - wciągnięta, 114, 115

- zmienna
 - zewnątrzna, 114
- logiczne, 29
 - przypisania, 46
 - puste, 45
 - składnia
 - kaskadowa, 150, 151
 - wyrażeniowa, 150, 151
 - zapytaniowe, 150, 156

X

- XML, 188
 - znacznik
 - c, 189
 - code, 190
 - example, 189
 - exception, 189
 - include, 190
 - list, 190
 - para, 190
 - param, 189
 - paramref, 190
 - permission, 189
 - remarks, 189
 - returns, 189
 - see, 190
 - seealso, 190
 - summary, 189

Z

- zapięczętowanie
 - implementacji, 79
- zapytanie, 140, 150, 153
 - zintegrowane, 139, *Patrz też:* LINQ
- zasada przypisań
 - oznaczonych, 39
- zdarzenie, 8, 108
 - akcesor, *Patrz:* akcesor
 - EventArgs, 107
 - implementacja, 109

- KeyEventArgs, 107
- MouseEventArgs, 107
- nadawca, *Patrz:* nadawca
- subskrybent, *Patrz:* subskrybent
- wirtualne, 77
- zestaw, 9
 - zaprzyjaźniony, 87
- Zip, *Patrz:* złączenie
 - suwakowe
 - złączenie, 155
 - suwakowe, 157
- zmienna, 13, 38, 45
 - deklaracja, 44
 - inicjalizacja, 44
 - lokalna, 38, 39, 52, 114
 - nazwa, 10
- znak
 - !, 48, 133, 134
 - !=, 29, 49, 85, 93, 132, 134
 - \$, 32
 - %, 24, 48, 133, 134
 - &, 26, 48, 49, 93, 133, 134, 182, 184
 - &&, 29, 49
 - &=, 49
 - (), 47, 48
 - () , 12
 - *, 12, 24, 48, 133, 134, 182, 184
 - */, 13
 - *=, 49
 - ., 12, 47
 - /, 24, 48, 133, 134
 - /*, 13
 - //, 13
 - /=, 49
 - ;, 12
 - ?, 130
 - ?., 48
 - ?:, 49
 - ??, 134
 - @, 11

znak	++, 25, 47, 48, 93, 133, 134	>, 29, 48, 93, 132, 134
[], 47		->, 47, 182, 184
^, 26, 49, 93, 133, 134	+=, 49, 93	>=, 29, 49, 93, 132
^=, 49	<, 29, 48, 93, 132, 134	>>, 26, 48, 133, 134
{, 12	<<, 26, 48, 133, 134	>>=, 49
, 26, 49, 93, 133, 134	<<=, 49	biała spacja, 34
, 29, 49	<=, 29, 48, 93, 132	ciąg, <i>Patrz:</i> ciąg
=, 49	=, 12, 46, 49, 93	znaków
}, 12	-=, 49, 93	interpunkcyjny
~, 26, 48, 93, 133, 134	==, 13, 29, 49, 85, 93, 131, 132, 134	języka C, 12
+, 24, 32, 48, 93, 133, 134	=>, 49	sterujący, 30

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

Przekonaj się, jaki jest C# — nowoczesny, prosty, efektywny!

C# został zaprojektowany jako obiektowy język programowania z kontrolą typów. C# w wersji 6.0 jest dojrzałą technologią, narzędziem, dzięki któremu można efektywnie pisać bezpieczny, przejrzysty i wydajny kod. Język ten z założenia ma być prosty w stosowaniu, niekiedy jednak programista potrzebuje pomocy: trzeba szybko coś sprawdzić, upewnić się co do przyjętych rozwiązań, przypomnieć sobie rzadziej używaną konstrukcję.

Niniejsza książka jest zwięzłym i praktycznym kompendium. Zawiera dokładnie to, co powinna – bez nudnych wywodów i niepotrzebnych dywagacji. Może posłużyć jako podręcznik do nauki C# lub jako poręczna ściągawka, pozwalająca na szybkie odnalezienie informacji. Jeśli tylko znasz podstawy programowania w Javie, C++ w poprzednich wersjach C# i chcesz bez większych problemów przystąpić do programowania w C# 6.0, trzymasz w ręku właściwą książkę.

Joseph Albahari – jest autorem kilku książek o programowaniu w języku C#. Jest twórcą LINQPada, popularnego narzędzia do prototypowania zapytań LINQ.

Ben Albahari pracował w firmie Microsoft, gdzie był szefem wielu ważnych projektów. Jest współtwórcą serwisu Auditionist, obsługującego wirtualne castingi dla aktorów w Wielkiej Brytanii.

W leksykonie przedstawiono:

- podstawy języka C#
- stosowanie technologii LINQ w pracy na kolekcjach danych
- wiązania dynamiczne i funkcje asynchroniczne
- wskaźniki, atrybuty, dyrektywy preprocesora i wiele innych zagadnień

Helion 

księgarnia Internetowa

<http://hellon.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

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

Sprawdź najnowsze promocje:
● <http://hellon.pl/promocje>
Książki najchętniej czytane:
● <http://hellon.pl/bestsellery>
Zamów informacje o nowościach:
● <http://hellon.pl/nowosci>

ślęgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-283-2446-6



9 788328 324466

Informatyka w najlepszym wydaniu

cena: 34,90 zł