

Kompendium C# zawsze pod ręką!

Leksykon kieszonkowy

C# 5.0



HELION

O'REILLY®

Joseph Albahari, Ben Albahari

Tytuł oryginału: C# 5.0 Pocket Reference: Instant Help for C# 5.0 Programmers

Tłumaczenie: Przemysław Szeremiota

ISBN: 978-83-246-6273-9

© 2013 Helion S.A.

Authorized Polish translation of the English edition C# 5.0 Pocket Reference,
First Edition ISBN 9781449320171 © 2012 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 jakiegokolwiek 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.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi 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/ch5lk3>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Leksykon kieszonkowy	5
Konwencje typograficzne	5
Korzystanie z przykładowych programów	6
Pierwszy program w C#	7
Składnia	10
System typów	13
Typy liczbowe	22
Typ wartości logicznych i operatory logiczne	29
Znaki i ciągi znaków	31
Tablice	34
Zmienne i parametry	38
Operatory i wyrażenia	46
Instrukcje	51
Przestrzenie nazw	58
Klasy	62
Dziedziczenie	72
Typ object	80
Struktury	84
Modyfikatory dostępu	85
Interfejsy	87
Typy wyliczeniowe	90
Typy zagnieżdżone	92
Uogólnienia	93
Delegaty	101
Zdarzenia	107
Wyrażenia lambda	113
Metody anonimowe	116
Wyjątki i instrukcja try	117
Enumeratory i iteratory	124
Typy z dopuszczalną wartością pustą	129

Przeciążanie operatorów	134
Metody rozszerzające	137
Typy anonimowe	139
LINQ	139
Wiązanie dynamiczne	163
Atrybuty	171
Atrybuty wywołania	175
Funkcje asynchroniczne	176
Wskaźniki i kod nienadzorowany	185
Dyrektywy preprocesora	189
Dokumentacja XML	191
O autorach	195
Skorowidz	197

Wiązanie dynamiczne

Wiązanie dynamiczne oznacza przesunięcie momentu *wiązania* — procesu ustalania typów, składowych i wywołań — od czasu kompilacji do czasu wykonania programu. Wiązanie dynamiczne zostało wprowadzone do języka C# w wersji 4.0; ma zastosowanie, kiedy *programista* wie, że pewna metoda, składowa czy operacja istnieje, ale *kompilator* nie ma o niej informacji. Do takich sytuacji dochodzi często w ramach interoperacji z językami dynamicznymi (jak IronPython) i obiektami COM, a także w sytuacjach typowych dla zastosowań mechanizmów refleksji.

Typ dynamiczny jest deklarowany z kontekstowym słowem kluczowym `dynamic`:

```
dynamic d = GetSomeObject();  
d.Quack();
```

Typ dynamiczny nakazuje kompilatorowi rozluźnić kontrolę typów; programista oczekuje, że w czasie wykonania zmienna typu `d` będzie posiadała metodę `Quack`. Jedyną trudność w tym, że w czasie kompilacji nie można tego potwierdzić. Ponieważ `d` jest zmienną typu dynamicznego, kompilator opóźni wiązanie wywołania metody `Quack` na rzecz `d` do czasu wykonania programu. Aby lepiej zrozumieć, co to oznacza, należałoby zdefiniować rozróżnienie pomiędzy *wiązaniem statycznym* i *dynamicznym*.

Wiązanie statyczne a wiązanie dynamiczne

Klasyycznym przykładem wiązania jest odwzorowanie nazwy występującej w kompilowanym wyrażeniu na konkretną metodę czy składową. Na przykład w poniższym wyrażeniu kompilator musi odszukać implementację metody o nazwie `Quack`:

```
d.Quack();
```

Załóżmy, że statyczny typ `d` to `Duck`:

```
Duck d = ...  
d.Quack();
```

W najprostszym przypadku kompilator dokonuje wiązania poprzez wyszukanie bezparametrowej metody `Quack` w klasie `Duck`. Jeśli to się nie powiedzie, kompilator rozszerzy poszukiwania na metody z parametrami opcjonalnymi, metody typów bazowych, a wreszcie metody rozszerzające, które przyjmują zmienną typu `Duck` w miejsce pierwszego argumentu wywołania. Jeśli i to nie doprowadzi do dopasowania

wywołania do metody, kompilator zgłosi błąd kompilacji. Jak widać, niezależnie od tego, jaka metoda zostanie ostatecznie wybrana do realizacji wywołania, wybór ten jest dokonywany przez kompilator, a samo dopasowanie wywołania i metody bazuje wyłącznie na informacji dostępnej statycznie, to znaczy na widocznych w kodzie definicjach typów operandów (tutaj `d`). Tyle o *wiązaniu statycznym*.

Zmieńmy teraz statyczny typ zmiennej `d` na typ `object`:

```
object d = ...
d.Quack();
```

Wywołanie `Quack` spowoduje teraz błąd kompilacji, ponieważ choć wartość przechowywana w `d` może zawierać metodę `Quack`, kompilator nie może jej znać, gdyż jedyną informacją, jaką dysponuje, jest najogólniejszy z możliwych typ zmiennej `object`. Co innego, jeśli typ `d` zostanie określony jako `dynamic`:

```
dynamic d = ...
d.Quack();
```

Typ dynamiczny jest trochę jak typ `object`, to znaczy równie mało mówi o zmiennej `d`. Różnica polega na tym, że typ dynamiczny może być używany w sposób nieweryfikowalny w czasie kompilacji. Operacje na obiekcie deklarowanym jako `dynamic` będą rozstrzygane w oparciu o typ obiektu ustalony w czasie wykonania programu, a nie typ z czasu kompilacji. Kompilator, napotkawszy wyrażenie wiązane dynamicznie (czyli wyrażenie, w którym występuje dowolna wartość typu dynamicznego), ogranicza się jedynie do takiego obrobienia wyrażenia, żeby wiązanie mogło zostać rozstrzygnięte w czasie wykonania programu.

W czasie wykonania, jeśli okaże się, że obiekt dynamiczny implementuje interfejs `IDynamicMetaObjectProvider`, to właśnie ten interfejs zostanie wykorzystany do rozstrzygnięcia dopasowania; w innym przypadku środowisko wykonawcze rozstrzygnie wywołanie analogicznie, jak zrobiłby to kompilator w czasie kompilacji, to znaczy przeszuka aktualny typ obiektu, jego typy bazowe oraz metody rozszerzające. Rozstrzyganie w oparciu o implementację interfejsu `IDynamicMetaObjectProvider` nazywamy *wiązaniem ze wskazania* albo *wiązaniem niestandardowym* (ang. *custom binding*), a rozstrzyganie w oparciu o dostępną w czasie wykonania wiedzę o typie i jego typach bazowych nazywamy *wiązaniem według reguł* albo *wiązaniem językowym* (ang. *language binding*).

Wiązanie ze wskazania

Wiązanie ze wskazania dotyczy przypadku, kiedy obiekt typu dynamicznego implementuje interfejs `IDynamicMetaObjectProvider` (IDMOP). Interfejs ten można co prawda implementować w typach pisanych w języku C#, ale najczęściej dotyczy to obiektów pozyskiwanych z innych języków dynamicznych z rodziny .NET, operujących w ramach środowiska wykonawczego Dynamic Language Runtime (DLR) (jak IronPython czy IronRuby). Obiekty pochodzące z tych języków niejawnie implementują interfejs IDMOP jako sposób bezpośredniego kontrolowania znaczenia operacji, które będą na tych obiektach wykonywane. Oto prosty przykład:

```
using System;
using System.Dynamic;

public class Test
{
    static void Main()
    {
        dynamic d = new Duck();
        d.Quack(); //wywołano Quack
        d.Waddle(); //wywołano Waddle
    }
}

public class Duck : DynamicObject
{
    public override bool TryInvokeMember (
        InvokeMemberBinder binder, object[] args,
        out object result)
    {
        Console.WriteLine ("wywołano " + binder.Name);
        result = null;
        return true;
    }
}
```

Klasa `Duck` w istocie nie posiada metody `Quack`, a jedynie wskazanie wiązania, dzięki któremu przechwytuje i interpretuje wywołania wszystkich metod na rzecz obiektów tej klasy.

Wiązania ze wskazania są omawiane szerzej w 20. rozdziale książki *C# 5.0 in a Nutshell*.

Wiązanie językowe

Z wiązaniem językowym mamy do czynienia w przypadku obiektów dynamicznych, które nie implementują interfejsu `IDynamicMetaObjectProvider`. Wiązanie językowe stosuje się przede wszystkim do ob-

chodzenia niedogodności niedoskonale zaprojektowanych typów, a także niektórych ograniczeń właściwych systemowi typów platformy .NET. Typowym problemem jest na przykład brak jednolitego interfejsu typów liczbowych (wiemy już, że wiązanie dynamiczne dotyczy metod; dopowiedzmy, że dotyczy również operatorów):

```
static dynamic Mean (dynamic x, dynamic y)
{
    return (x + y) / 2;
}

static void Main()
{
    int x = 3, y = 4;
    Console.WriteLine (Mean (x, y));
}
```

W takich przypadkach zalety wiązania dynamicznego są oczywiste, bo nie trzeba powielać kodu wspólnych operacji obliczeniowych dla każdego z rozmaitych typów liczbowych. Wadą jest natomiast utrata statycznej kontroli typów, a więc i ryzyko zamiany błędów kompilacji na wyjątki czasu wykonania.

Uwaga

Wiązanie dynamiczne oznacza osłabienie kontroli typów, ale tylko statycznej — kontrola dynamiczna wciąż jest obecna. Inaczej niż w mechanizmach refleksji, wiązanie dynamiczne nie pozwala np. na ominięcie reguł widoczności składowych.

Dynamiczne wiązanie językowe celowo ma możliwie blisko odwzorowywać wiązanie statyczne (a więc działa tak, jak działałby kompilator, gdyby tylko typ dynamiczny był mu znany statycznie). Zachowanie programu z poprzedniego przykładu byłoby identyczne, gdybyśmy jawnie oprogramowali metodę `Mean` do operowania na typie `int`. Najbardziej znamienna różnica pomiędzy wiązaniem dynamicznym i statycznym dotyczy metod rozszerzających; będzie o tym mowa w punkcie „Funkcje niedynamiczne”.

Uwaga

Wiązanie dynamiczne wiąże się z narzutem wydajności wykonania programu, ale dzięki mechanizmom cachowania DLR powtarzające się wykonania tych samych wyrażeń wiązanych dynamicznie są skutecznie optymalizowane, co pozwala stosunkowo wydajnie wykonywać wywołania dynamiczne na przykład w pętlach. Optymalizacja redukuje narzut rozstrzygnięcia prostego wyrażenia dynamicznego do około 100 ns (mowa o współczesnym komputerze).

Wyjątek `RuntimeBinderException`

Jeśli składowej nie uda się związać dynamicznie, środowisko wykonawcze zgłosi wyjątek `RuntimeBinderException`, będący odpowiednikiem błędu czasu kompilacji, ale wykrytym dopiero w czasie wykonania:

```
dynamic d = 5;
d.Hello(); //wywołanie zgłosi wyjątek RuntimeBinderException
```

Wyjątek zgłoszony w powyższym kodzie oznacza, że typ dynamiczny (tu `int`) nie posiada metody `Hello`.

Reprezentacja typu dynamicznego

Typ dynamiczny cechuje się znacznym podobieństwem do typu `object` — środowisko traktuje na przykład poniższe porównanie jako prawdziwe:

```
typeof (dynamic) == typeof (object)
```

Zasada ta rozciąga się również na typy konkretyzowane typem dynamicznym i typy tablicowe:

```
typeof (List<dynamic>) == typeof (List<object>)
typeof (dynamic[]) == typeof (object[])
```

Tak jak w przypadku referencji `object`, referencja `dynamic` może odnosić się do obiektu dowolnego typu (za wyjątkiem typów wskaźnikowych):

```
dynamic x = "ahoj";
Console.WriteLine (x.GetType().Name); //String
```

```
x = 123; //nie ma błędu (choć to ta sama zmienna)
Console.WriteLine (x.GetType().Name); //Int32
```

Pomiędzy referencjami `object` i referencjami `dynamic` strukturalnie nie ma żadnej różnicy. Referencja `dynamic` pozwala po prostu na dynamiczne wiązanie operacji na obiekcie, do którego się odnosi. Można nawet przekonwertować obiekt `object` na typ `dynamic` i następnie wykonać na nim dowolną operację dynamiczną:

```
object o = new System.Text.StringBuilder();
dynamic d = o;
d.Append ("ahoj");
Console.WriteLine (o); //ahoj
```

Konwersje typów dynamicznych

Typ dynamiczny daje się niejawnie skonwertować na dowolny inny typ, a także dowolny inny typ można niejawnie konwertować na typ dynamiczny. Aby jednak konwersja była skuteczna, właściwy typ obiektu dynamicznego musi faktycznie dać się skonwertować na docelowy typ statyczny.

Poniższy kod spowoduje wyjątek `RuntimeBinderException`, ponieważ typ `int` nie daje się niejawnie konwertować na typ `short`:

```
int i = 7;
dynamic d = i;
long l = d; // OK — niejawna konwersja działa
short j = d; // wyjątek RuntimeBinderException
```

var a dynamic

Typy `var` i `dynamic` wydają się podobne, ale w istocie zasadniczo się różnią:

- `var` oznacza: „niech kompilator określi faktyczny typ”;
- `dynamic` oznacza: „niech środowisko wykonawcze określi faktyczny typ”.

Spójrzmy:

```
dynamic x = "ahoj"; // statyczny typ to dynamic
var y = "ahoj"; // statyczny typ to string
int i = x; // wyjątek wykonania
int j = y; // błąd kompilacji
```

Wyrażenia dynamiczne

Wiązanie dynamiczne może dotyczyć pól, właściwości, metod, zdarzeń, konstruktorów, indeksów, operatorów i konwersji.

Nie można pobrać wyniku wyrażenia dynamicznego z typem zwracanym `void`; podobnie jest w przypadku wyrażeń statycznych. Różnica dotyczy momentu wystąpienia błędu (tutaj w czasie wykonania programu).

Wyrażenia dynamiczne to najczęściej takie wyrażenia, w których uczestniczą dynamiczne operandy, a to dlatego, że najczęściej niedostępność statycznej informacji o typie daje efekt kaskadowy:

```
dynamic x = 2;
var y = x * 3; // statyczny typ y to dynamic
```

Od tej reguły jest kilka oczywistych wyjątków. Po pierwsze, rzutowanie typu dynamicznego na typ statyczny daje wyrażenie statyczne. Po drugie, wywołanie konstruktora zawsze daje wyrażenie statyczne, nawet jeśli argumenty wywołania mają typy dynamiczne.

Istnieje też kilka przypadków brzegowych, w których wyrażenie zawierające argument dynamiczny samo jest statyczne; zaliczymy tu przekazanie indeksu do tablicy i wyrażenia tworzenia delegatów.

Rozstrzygnięcie przeciążeń składowych dynamicznych

Klasycznym przykładem użycia wiązania dynamicznego jest dynamiczny *odbiornik komunikatów*, to znaczy obiekt dynamiczny jako podmiot dynamicznego wywołania metody:

```
dynamic x = ...;
x.Foo (123); //x "odbiera" wywołania
```

Jednak zakres zastosowań typów dynamicznych jest szerszy i obejmuje także dynamiczne wiązanie argumentów wywołania metody, gdzie rozstrzygnięcie wywołania z dynamicznymi argumentami jest przesuwane z czasu kompilacji do czasu wykonania:

```
class Program
{
    static void Foo (int x) { Console.WriteLine ("1"); }
    static void Foo (string x) { Console.WriteLine ("2"); }
    static void Main()
    {
        dynamic x = 5;
        dynamic y = "arbuz";

        Foo (x); //1
        Foo (y); //2
    }
}
```

Rozstrzygnięcie przeciążenia w czasie wykonania jest również określane mianem wielorozprowadzania albo *multimetody* (ang. *multiple dispatch*), a stosuje się je między innymi w implementacjach wzorca projektowego Wizytator.

W przypadku dynamicznych argumentów wywołania metody (ale bez dynamicznego podmiotu wywołania) kompilator może przeprowadzić podstawowe sprawdziany skuteczności wywołania dynamicznego: sprawdzane jest więc choćby istnienie funkcji z odpowiednią nazwą i liczbą parametrów. Jeśli takiej metody nie ma, wiadomo, że rozstrzygnięcie dynamiczne również będzie nieudane, więc kompilator zgłasza błąd kompilacji.

W przypadku wywołań z argumentami typów dynamicznych oraz statycznych ostateczny wybór metody będzie odzwierciedlał mieszaninę decyzji podjętych statycznie i dynamicznie:

```
static void X(object x, object y) {Console.Write("oo");}
static void X(object x, string y) {Console.Write("os");}
static void X(string x, object y) {Console.Write("so");}
static void X(string x, string y) {Console.Write("ss");}
static void Main()
{
    object o = "ahoj";
    dynamic d = "pa, pa";
    X(o, d); //os
}
```

Wywołanie $X(o,d)$ jest wiązane dynamicznie, ponieważ jeden z jego argumentów ma typ `dynamic`. Ale skoro typ argumentu `o` jest statyczny, już w czasie kompilacji wiadomo, że w grę wchodzi jedynie dwa pierwsze przeciążenia X . Ostatecznie wybór padnie na drugie z nich, a to ze względu na dokładnie dopasowany statyczny typ `o` i dokładnie dopasowany dynamiczny typ `d`. Innymi słowy, kompilator usiłuje maksymalnie wykorzystać statyczną informację o typach nawet w przypadku wyrażen dynamicznych.

Funkcje niedynamiczne

Niektórych funkcji nie można wywołać dynamicznie. Dotyczy to:

- metod rozszerzających (ze składnią metod rozszerzających),
- dowolnych składowych interfejsu (w przypadku wywołań przez interfejs),
- składowych typów bazowych przykrytych typami pochodnymi.

Niemożność wynika z tego, że wiązanie dynamiczne potrzebuje dwóch informacji: nazwy metody do wywołania oraz obiektu, na rzecz którego ma się odbyć wywołanie. Ale w powyższych przypadkach uczestniczy *dotatkowy typ*, znany wyłącznie w czasie kompilacji. Język C# w wersji 5.0 nie daje możliwości dynamicznego określenia dodatkowego typu.

W przypadku wywołania metody rozszerzającej tym dodatkowym typem jest klasa rozszerzająca, wybierana niejawnie na bazie dyrektyw `using` widocznych w kodzie źródłowym (a więc widocznych tylko w czasie kompilacji). Przy wywołaniach za pośrednictwem interfejsu dodatkowy typ jest komunikowany jawną albo niejawną konwersją

(w przypadku implementacji jawnej nie da się wywołać składowej bez rzutowania na typ interfejsu). Tak samo w przypadku przykrywania składowych klasy bazowej: dodatkowy typ musi być określony albo poprzez rzutowanie, albo poprzez słowo kluczowe `base`, a oba są widoczne wyłącznie w czasie kompilacji.

A

abstract class, *Patrz:* klasa
abstrakcyjna
agregator zadań, 183
aplikacja, 9, 10
argument, 8
nazwany, 44
typowy, 94
assembly, *Patrz:* zestaw
atrybut, 171
CLSCompliant, 173
obiekt docelowy, 172
odwołania w czasie wykonania,
174
własny, 173
wywołania, 175
XmlElementAttribute, 172
attribute target, *Patrz:* atrybut obiekt
docelowy

B

biała spacja, 34
biblioteka, 9, 10
blok
catch, 117, 118, 120
finally, 117, 119, 120
instrukcji, *Patrz:* instrukcja blok,
Patrz: instrukcja blok
try, 117, 118, 120
błąd wejścia-wyjścia, 120
boxing, *Patrz:* pakowanie
broadcaster, *Patrz:* nadawca

C

callback, *Patrz:* wywołanie zwrotne
caller info attributes, *Patrz:* atrybut
wywołania

captured variable, *Patrz:* zmienna
wciągnięta
cast, *Patrz:* rzutowanie
ciąg znaków, *Patrz:* znak ciąg
closure, *Patrz:* domknięcie
CLR, 22, 82, 177
CLS, 173
Common Language Runtime,
Patrz: CLR
Common Language Specification,
Patrz: CLS
conditional operator, *Patrz:* operator
warunkowy
constraint, *Patrz:* ograniczenie
constructor, *Patrz:* konstruktor

D

dane
składowe, 15
statyczne, 99
wejściowe, 8, 63
wyjściowe, 8
deferred execution, *Patrz:* operator
wykonanie opóźnione
delegat, 18, 96, 101, 102
Action, 105, 114
Func, 105, 114
instancja, 101, 102
metoda instancji, 104
pole prywatne, 112
typ, 101, 102
uogólnienie, 100
uogólniony, 105, 107, 114
wielokrotny, 103, 104
zmienna, 102
delegatów, 184
domknięcie, 114, 115
dostępność, 85

downcasting, *Patrz:* referencja
rzutowanie w dół
dynamic type checking, *Patrz:* typ
kontrola dynamiczna
dyrektywa
 preprocesora, 189, 190, 191
 using, 60
 warunkowa, 189
dziedziczenie, 72, 73, 78, 79, 84, 87,
88, 93

E

element, 140
enumerator, 124, 125
escape sequence, *Patrz:* znak sterujący
event, *Patrz:* zdarzenie

F

funkcja, 51
finalizator, 8, 62, 70, 84
finalizer, *Patrz:* finalizator
fluent syntax, *Patrz:* składnia
 kaskadowa
fully qualified type name, *Patrz:* typ
 nazwa w pełni kwalifikowana
funkcja, 8
 asynchroniczna, 176, 180, 181, 183
 niedynamiczna, 170
 przesłonięta, 75
 składowa, 15
 wirtualna, 75
 wysokopoziomowa, 7
 wywołanie, 40

G

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

H

hermetyzacja, 85

I

identyfikator, 10, 11
IL, 61
iloczyn logiczny, 30

indekser, 8, 62, 68, 87, 95, 168
 wirtualny, 75

indexer, *Patrz:* indekser
inferencja, 23

instancja
 konwersja, *Patrz:* konwersja
 składowa, 16

instantiation, *Patrz:* konkretyzacja

instrukcja, 7, 51
 blok, 7, 12, 51
 break, 57
 continue, 57
 deklaracji, 51
 fixed, 186
 foreach, 35, 125
 goto, 57, 58
 if, 52
 zagnieżdżona, 53, 54

if..else, 53
iteracyjna, 55
pętli, *Patrz:* pętla
return, 57, 58, 126, 128
skoku, 57
switch, 54
throw, 57
try, 117
using, 121, 125, 138
warunkowa, 52
wyboru, 54
wyrażeniowa, 52
yield, 127
yield break, 128
yield return, 126

interfejs, 18, 62, 87, 96

 API, 45
 deklaracja, 87
 IDisposable, 125
 IDMOP, 165
 IDynamicMetaObjectProvider, 165
 IDynamicMetaObjectProvider, 165
 IEnumerable, 35, 100, 139, 140
 IEnumerator, 87, 100
 implementacja, 89
 jawna, 88
 wirtualna, 89
 IQueryable, 140
 ISerializable, 89

- kowariantny, 100
 - rozszerzanie, 88
 - składowa, 170, *Patrz:* składowa interfejsu
 - System.
 - Collections.Generic.IEnumerator, 125
 - System.Collections.Generic.
 - ↳IEnumerator, 127
 - System.Collections.Generic.
 - ↳IEnumerator, 127
 - System.Collections.IEnumerable, 126, 127
 - System.Collections.IEnumerator, 125, 127
 - System.IDisposable, 121
 - Intermediate Language, *Patrz:* IL
 - IronPython, 163
 - iterator, 126, 127, 128
- J**
- jagged array, *Patrz:* tablica wyszczerbiona
 - język
 - dynamiczny, 163
 - IronPython, 163
 - pośredni, *Patrz:* IL
- K**
- klasa, 8, 16, 18, 62, 84, 87, 93, 96
 - abstrakcyjna, 76
 - atrybut, 62
 - bazowa, 62, 73, 76, 77, 80
 - BitArray, 29
 - dziedziczenie, *Patrz:* dziedziczenie
 - Enumerable, 150, 151
 - instancja
 - składowe, 16
 - konstruktor, 78
 - modyfikator, 62
 - pochodna, 73, 76, 77
 - statyczna, 16, 70
 - string, 68
 - System.Array, 35
 - System.Attribute, 171
 - System.EventArgs, 110
 - System.Exception, 123
 - System.Linq.Enumerable, 140, 146
 - System.Text.StringBuilder, 33
 - TaskCompletionSource, 181
 - klauzula
 - catch, 119
 - inicjalizacji, 56
 - iteracji, 56
 - klucz publiczny, 86
 - kod
 - eliminacja powtórzeń, 93
 - nienadzorowany, 185, 186
 - platformy, 109
 - uogólniony, 40
 - własny użytkowników, 109
 - XML, 191
 - źródłowy, 9, 10, 189
 - kolejka LIFO, *Patrz:* LIFO
 - kolizja nazw, 11
 - komentarz, 13
 - dokumentujący, 191, 192
 - jednowierszowy, 13
 - wielowierszowy, 13
 - kompilacja, 9, 10, 44, 71, 79, 163
 - /checked+, 26
 - /unsafe, 185
 - błąd, 27, 100, 190
 - czasu, 25
 - opcje, 26
 - ostrzeżenie, 190
 - kompilator, 9, 10, 13, 71, 112, 189
 - konkatenacja, 33
 - konkretyzacja, 15
 - konstruktor, 8, 16, 62, 63, 95, 168
 - bezparametrowy, 44, 64, 69, 79, 84, 85
 - klasy, 78
 - bazowej, 77
 - niejawny, 64
 - niepubliczny, 64
 - przeciążanie, 64
 - statyczny, 69, 70
 - kontekst
 - nienadzorowany, 180
 - synchronizacji, 179
 - unsafe, 39
 - kontrawariancja, 99, 101, 106, 107

- konwersja, 17, 73, 168
 - jawna, 17, 18, 24, 26, 130
 - liczbowa, 75
 - niejawna, 17, 18, 24, 130
 - numeryczna, 100
 - odpakowywania, 75, 100
 - przeciążanie, 136
 - typów dynamicznych, 168
 - użytkownika, 75, 100
- kowariancja, 99, 100, 101, 106, 107
- kropka, 12, 48, 59
- kwalifikator, 146, 149
 - global, 61
- kwantyfikator, 143

L

- lambda, 48, 51, 113, 114, 116, 178
 - asynchroniczne, 184
 - parametr, 113
 - zmienne
 - wciągnięte, 114, 115
 - zewnętrzne, 114
- lambda expression, *Patrz:* lambda
- Language Integrated Query, *Patrz:* LINQ
- lazy execution, *Patrz:* operator wykonanie leniwe
- liczba, 8
 - całkowita, 15, 18, 21, 24
 - 16-bitowa, 27
 - 8-bitowa, 27
 - rzeczywista, 23, 24
 - zaokrąglenie, 28
 - zmiennoprzecinkowa, 24
- LIFO, 80
- lift, *Patrz:* operator pożyczanie
- LINQ, 139
 - element, *Patrz:* element
 - sekwencja, *Patrz:* sekwencja
- LINQPad, 140
- lista, 139
- literal, 8, 12
 - dosłowny ciągów znaków, 32
 - liczb
 - całkowitych, 23
 - rzeczywistych, 23
 - liczbowy, 23

- przyrostek, 23
- typ, 23
- null, *Patrz:* null
- znakowy, 31

M

- makrodefinicja, 69
- mechanizm
 - asynchronicznego wywołania funkcji, 176
 - atrybutów, 171
 - cachowania, 166
 - jawnego implementowania składowej interfejsu, 88
 - odświeżania, 39, 71, 186
- metoda, 7, 8, 9, 15, 62, 63, 66, 87, 96, 168
 - Aggregate, 148
 - All, 149
 - anonimowa, 116, 117, 184
 - Any, 149
 - Append, 33
 - AsEnumerable, 149
 - AsQueryable, 149
 - asynchroniczna, 177, 181, 182
 - Average, 148
 - BinarySearch, 35
 - Cast, 149, 162
 - Combine.System.Delegate, 104
 - CompareTo, 33
 - Concat, 148
 - Contains, 33, 149
 - Copy, 35
 - Count, 148
 - CreateInstance, 35
 - częściowa, 71, 72
 - DefaultIfEmpty, 148
 - deklaracja, 40
 - Dispose, 121
 - Distinct, 147
 - double.IsNaN, 28
 - ElementAt, 148
 - ElementAtOrDefault, 148
 - Empty, 149
 - EndsWith, 33
 - Except, 148
 - Finalize, 71

- Find, 35
- Find LastIndex, 35
- FindIndex, 35
- First, 148
- FirstOrDefault, 148
- float.IsNaN, 28
- GetEnumerator, 125
- GetLength, 36
- GetType, 82
- GetValue, 35
- GroupBy, 148, 160
- GroupJoin, 147, 156, 157, 158
- IndexOf, 33, 35
- Insert, 33, 34
- instancji, 138
- Intersect, 148
- Join, 34, 147, 156, 158
- Last, 148
- LastIndexOf, 33, 35
- LastOrDefault, 148
- LongCount, 148
- Max, 148
- Min, 148
- MoveNext, 144
- nienazwana, 184
- object.Equals, 28, 83, 84
- object.GetHashCode, 83, 84
- object.ReferenceEquals, 83
- object.ToString, 84
- OfType, 149, 162
- OrderBy, 147, 150, 159
- OrderByDescending, 147
- PadLeft, 34
- PadRight, 34
- parametr, *Patrz*: parametr metody
- przeciążanie, 63, 79
- przesłaniana, 76
- publiczna, 44
- Range, 149
- rekurencyjna, 39
- Remove, 33, 34
- Remove.System.Delegate, 104
- Repeat, 149
- Replace, 33
- Reverse, 147
- rozszerzająca, 137, 138, 150, 151, 170
 - wywołanie kaskadowe, 138
- Select, 141, 147, 150
- SelectMany, 147, 155
- SequenceEqual, 149
- SetValue, 35
- Single, 143, 148
- SingleOrDefault, 143, 148
- Skip, 147
- SkipWhile, 147
- Sort, 35
- Split, 34
- StartsWith, 33
- Substring, 34
- Sum, 148
- sygnatura, 63
- Take, 147
- TakeWhile, 147
- Task.Run, 177
- ThenBy, 147, 159
- ThenByDescending, 147
- ToArray, 149
- ToDictionary, 149
- ToList, 149
- ToLookup, 149
- ToLower, 34
- ToString, 33
- ToUpper, 34
- Trim, 34
- TrimEnd, 34
- TrimStart, 34
- Union, 148
- uogólniona, 95, 96
- WhenAll, 183
- Where, 141, 147, 150
- wirtualna, 75, 76, 83
- wtyczka, 102
- Zip, 147, 159
- modyfikator
 - async, 184
 - dostępu, 85
 - internal, 85
 - new, 77
 - out, 41, 43, 44, 100
 - override, 75
 - params, 43, 44
 - private, 85
 - protected, 85
 - protected internal, 85
 - public, 85

modyfikator
 ref, 41, 42, 44, 171
 this, 137
 unsafe, 179
 virtual, 171
multimetoda, 169
multiple dispatch, *Patrz:* multimetoda

N

nadawca, 107, 108
namespace, *Patrz:* przestrzeń nazw
NaN, 27
nawias klamrowy, 12
negacja wartości wyrażenia
 logicznego, 30
nested type, *Patrz:* typ zagnieżdżony
not a number, *Patrz:* NaN
null, 20, 40, 48, 75, 97, 129, 133
null coalescing operator, *Patrz:* znak ??
nullable type, 21, *Patrz:* typ
 z dopuszczalną wartością pustą

O

obiekt, 39
 char, 31
 COM, 163
 docelowy atrybutu, 172
 dynamiczny, 165, 169
 inicjalizator, 65
 nadzorowany, 186
 przeliczalny, 57, 125
 referencja, 39
 System.Type, 82
 tworzenie, 39
 zwalnianie ze sterty, 39
obsługa
 błędów, 117
 wyjątków, 118, 119
odbiornik komunikatów, 169
odpakowywanie, 80, 81
ograniczenie, 97, 98
operacja
 arytmetyczna, 25
 bitowa, 30
 dzielenia, 25
 iterowania, 125
 na liczbach całkowitych, 25
 przypisania, 13

operand, 46
operator, 8, 12, 46, 62, 95, 168
 agregacji, 143, 146, 148
 All, 143
 alternatywy dla null, 48
 Any, 143
 arytmetyczny, 25, 92
 as, 75
 Average, 143
 bitowy, 26, 30, 91, 92
 checked, 26
 Concat, 144
 Contains, 143
 Count, 143
 dekrementacji, 25, 26
 dodawania, 92
 dostępu do składowej przez
 wskaźnik, 185, 187
 dwuargumentowy, 46
 eksportu, 146, 149
 ElementAt, 142
 elementowy, 146, 148
 filtrujący, 146, 147
 First, 142
 funkcja, 134
 generujący, 146, 149
 główny, 48, 49
 grupowania, 146, 148
 importu, 146, 149
 indeksowania, 33
 inkrementacji, 25, 26
 is, 75
 jednoargumentowy, 46, 49
 konkatenacji, 33
 konwersji, 146, 149
 Last, 142
 logiczny, 30
 łączenia, 146, 147, 156
 łączność, 47
 lewostronna, 48
 prawostronna, 48
 Max, 143
 Min, 143
 mnożenia, 12, 26
 new, 16
 obliczenia nadzorowanego, 26
 odwołania, 12

OrderBy, 150
pierwszeństwo, 47
pierwszorzędny, 46
pobrania adresu, 185
porównania, 13, 28, 29, 83, 92, 132
 przeciążanie, 135
porządkujący, 146, 147
pożyczanie, 131, 132
priorytet, 47, 48
projekcji, 146, 147
przeciążanie, 48, 84, 134
przedrostkowy, 25
przypisania, 47, 48, 51
przyrostkowy, 25
relacji, 29, 30, 33, 132
 przeciążanie, 135
reszty z dzielenia, 25
Reverse, 142
Select, 141, 150
SequenceEquals, 143
Single, 142
sizeof, 92
Skip, 142
Take, 142
trójargumentowy, 46
trójwartościowy, 48, *Patrz:*
 operator warunkowy
typeof, 96
typu wyliczeniowego, 92
unchecked, 26
Union, 144
warunkowy, 30, 48
Where, 141, 150
wykonanie leniwe, 144
wykonanie opóźnione, 144
wyłuskania, 185, 188
wywołania metody, 46
z przypisaniem, 47
zapytania, 140, 146
 kaskadowego, 149
zbiorów, 144, 146, 148
outer variable, *Patrz:* zmienna
 zewnętrzna
overflow, *Patrz:* przepełnienie
 zakresu
override function, *Patrz:* funkcja
 przesłonięta

P

pakowanie, 80, 81, 93
parametr, 8, 40, 43
 metody, 63
 nazwany, 172
 opcjonalny, 40, 43, 44, 45
 pozycyjny, 172
 przekazywany przez referencję,
 38, 42
 przekazywany przez wartość, 38,
 41, 42
 referencyjny, 63
 typowy, 93, 95
 deklarowanie, 96
 kowariantny, 100
 wartość domyślna, 97
 wyjściowy, 38, 63
partially qualified name, *Patrz:*
 przestrzeń nazw częściowa
 kwalifikacja nazw
pętla, 55, 115
 do-while, 55
 for, 34, 55, 56
 foreach, 55, 57, 116
 while, 55
podklasa, *Patrz:* klasa pochodna
pole, 40, 62, 66, 95, 168
 inicjalizacja, 62
 instancji, 38
 statyczne, 38
polimorfizm, 73, 106
preprocesor, 189, 190, 191
primary operators, *Patrz:* operator
 pierwszorzędny
primitive type, *Patrz:* typ:prosty
programowanie
 asynchroniczne, 183
programowanie asynchroniczne, 176
property, *Patrz:* właściwość
protokół, 101
przeciążanie, 96
przepełnienie zakresu, 25
przestrzeń nazw, 9, 58, 59, 60
 częściowa kwalifikacja nazw, 60
 deklaracja, 61
 globalna, 59
 hierarchia, 59

przestrzeń nazw
import, 60
importowanie, 61, 138
przesłanianie nazw, 61
System, 14, 21
System.Collections, 29, 35, 87
System.Security.Cryptography, 58

Q

query expression, *Patrz*; wyrażenie
zapytaniowe

R

rectangular array, *Patrz*: tablica
regularna
refaktoryzacja, 86
reference type, *Patrz*: typ
referencyjny
referencja, 19, 29, 39, 83
konwersja, 73, 75
niejawna, 100
polimorfizm, *Patrz*: polimorfizm
pusta, 30, 129
rzutowanie, 73
w dół, 73, 74, 94
w górę, 73, 74
this, 65
rekurencja, 39
relacja, 50
rethrow, *Patrz*: wyjątek ponowne
zgłoszenie
rozgłaszanie, 108, 117
rzutowanie, 18, 81, 83, 93, 185, *Patrz*
też: referencja rzutowanie
jawne, 81, 91, 94, 131

S

sealing, *Patrz*: zapieczętowanie
sekwencja, 140
konkatenacja, 144
lokalna, 140
wejściowa, 140, 141
wyjściowa, 140, 141
short-circuit, *Patrz*: zasada
skróconego obliczania
składnia
kaskadowa, 151, 152
wyrażeń zapytaniowych, 151, 152

składowa
abstrakcyjna, 76, 87
instancji, *Patrz*: instancja
składowa
interfejsu, 87, 89, 170
klasy, 17, 62, 76
przestarzała, 171
statyczna, 16, 39
typu
wyliczeniowego, 90, 91
bazowego, 170
wirtualna, 76, 84
zapieczętowana, 89
słowa kluczowe
async, 178
await, 178, 179, 180
słownik, 35
słowo kluczowe, 10, 11
async, 176
await, 176
base, 77, 79
class, 62
default, 40
delegate, 102, 116
descending, 160
event, 108, 109
explicit, 134
fixed, 187
get, 66, 67
implicit, 134
internal, 59
into, 154, 158
kontekstowe, 12
let, 153
namespace, 59
orderby, 159
private, 59
public, 17, 59
sealed, 77
set, 66, 67
sizeof, 92
stackalloc, 187
static, 16, 70
struct, 18
this, 64, 77, 79
throw, 57
typeof, 96

- unsafe, 185
- var, 45, 46
- virtual, 75, 89
- stała, 39, 46, 69
 - deklaracja, 51
 - lokalna, 52
 - typ, 69
 - typu referencyjnego, 19
 - typu wartościowego, 18
- statement, *Patrz:* instrukcja
- sterta, 38, 39, 186
- stos, 38, 80, 95, 187
- strong name assembly, *Patrz:* zestaw zaprzyjaźniony podpisany
- struct, *Patrz:* struktura
- struktura, 84, 85, 87, 93, 96
 - Nullable, 130, 131
- subclass, *Patrz:* klasa pochodna
- subscriber, *Patrz:* subskrybent
- subskrybent, 107, 108
- suma logiczna, 30, 50
- symbol specjalny, 31, 32
- system generowania dokumentacji, 13

Ś

- średnik, *Patrz:* znak ;

T

- tablica, 8, 18, 34, 43, 57, 139, 186, 187
 - ciągów znaków, 8
 - deklaracja, 34
 - dynamiczna, 35
 - element, 38, 40
 - typu wartościowego, 84
 - inicjalizacja, 36, 37, 38
 - jako argument wywołania funkcji., 38
 - liczba wymiarów, 35
 - odwołanie
 - indeksowanie, 34
 - prostokątna, *Patrz:* tablica regularna
 - regularna, 36
 - rozmiar, 35, 36, 37
 - tablic, *Patrz:* tablica wyszczerbiona
 - wartości typu bool, 29

- wielowymiarowa, 36
- wyrażenia inicjalizacji, 35
- wyszczerbiona, 36, 37
- zagnieżdżona, *Patrz:* tablica wyszczerbiona
- task combinator, *Patrz:* agregator zadań
- typ, 9, 13
 - anonimowy, 139
 - bool, 14, 18, 21, 29, 40, 133
 - byte, 27
 - całkowity, 7, 8, 24
 - char, 18, 21, 31, 40
 - ciągu znaków, 21
 - częściowy, 71
 - decimal, 23, 24, 28
 - delegatu, 104
 - dookreślony, *Patrz:* typ zamknięty
 - double, 23, 24, 28
 - dynamic, 168
 - dynamiczny, 163, 165, 167
 - konwersja, 168
 - enum, 18, *Patrz:* typ wyliczeniowy
 - false, 40
 - float, 23, 24, 28
 - instancja, 15, 18
 - int, 14, 15, 18, 22, 27
 - konkretyzowanie, 16
 - kontrola
 - dynamiczna, 74, 82
 - statyczna, 82
 - liczbowy, 18, 21, 22, 40
 - logiczny, 18, 21, 29, 40
 - long, 18, 22, 23, 27
 - model refleksji, 82
 - nazwa
 - w pełni kwalifikowana, 59
 - kwalifikowana, 61
 - niedookreślony, *Patrz:* typ otwarty
 - obiektowy, 21
 - object, 21, 80
 - otwarty, 94
 - pochodny, 98
 - predefiniowany
 - wartościowy, 22
 - predefiniowany, 13, 15, 21, 40

prosty, 14, 22
przestarzały, 171
referencyjny, 18, 19, 21, 29, 32, 36,
40, 42, 80, 84, 97, 129
reprezentacja, 82
RSA, 58
sbyte, 27
short, 18, 27
statyczny, 46
string, 14, 21, 32, 33
struct, 18
strukturowy, 18
synonim, 62
uint, 23
ulong, 23
uogólniony, 93, 95, 96
niezwiązany, 96
pochodne, 98
ushort, 27
var, 168
void, 8
wartościowy, 18, 21, 32, 42, 84,
97, 185
instancja, 39
wbudowany, *Patrz:* typ
predefiniowany
własny, 14, 15, 40
tworzenie, 18
wnioskowany, 23
wskaźnikowy, 185
wyliczeniowy, 18, 40, 90
konwersja, 91
operator, 92
z dopuszczalną wartością pustą,
130, 131, 133
zagnieżdżony, 62, 92, 93
zamknięty, 94
złożony, 14
zmiennoprzecinkowy, 23, 24, 27, 28
znakowy, 18, 21, 40
typ referencyjny
instancja, 39
type argument, *Patrz:* argument
typowy
type parameter, *Patrz:* parametr
typowy

U

ukośnik, 13, 49, 132, 134
lewy, 31, 32
unboxing, *Patrz:* odpakowywanie
uogólnienie, 93, 96, 97, 104
delegatów, 100
kowariantne, 99
ograniczenia, 97
upcasting, *Patrz:* referencja
rzutowanie w górę

V

value type, *Patrz:* typ wartościowy
verbatim string literal, *Patrz:* literal
dosłowny ciągów znaków
void, 63, 188
void expression, *Patrz:* wyrażenie
puste

W

wartość
+∞, 27
-0, 27
-∞, 27
domyślna, 40, 43
Epsilon, 27
liczbowa, *Patrz:* literal
logiczna, 29
MaxValue, 27
MinValue, 27
NaN, *Patrz:* NaN
nieliczbowa, *Patrz:* NaN
pusta, *Patrz:* null
zwracana, 8
whitespace, *Patrz:* biała spacja
wiązanie, 163
dynamiczne, 163, 166, 168
językowe, 165
ze wskazania, 165
widoczność, *Patrz:* dostępność
wielorozprowadzanie, 169
właściwość, 8, 62, 66, 67, 87, 95, 168
akcesor, 66, 67
automatyczna, 67
wirtualny, 75
wskaźnik, 185
beztypowy, 188

- współbieżność, 176
 - wyjątek, 70
 - DivideByZeroException, 118
 - IndexOutOfRangeException, 35
 - InvalidCastException, 81
 - InvalidOperationException, 131
 - obsługa, 118, 119
 - odwołania do pustej referencji, 30
 - ponowne zgłoszenie, 117
 - RuntimeBinderException, 167, 168
 - System.ArgumentException, 123
 - System.ArgumentNullException, 124
 - System.ArgumentOutOfRangeException
 - ↳Exception, 124
 - System.InvalidOperationException, 124
 - System.NotImplementedException, 124
 - System.NotSupportedException, 124
 - System.ObjectDisposedException, 124
 - zgłaszanie, 122
 - ponowne, 122
 - wyliczenie, *Patrz:* typ wyliczeniowy
 - wyrazenie, 46
 - dynamiczne, 168
 - lambda, *Patrz:* lambda
 - logiczne, 56
 - przypisania, *Patrz:* operator przypisania
 - puste, 46
 - stałowartościowe, 44
 - zapytaniowe, 151, 152
 - kompilacja, 152
 - wywołanie
 - zwrotne, 102
- X**
- XML, 191, 192
- Z**
- zadania agregator, 183
 - zapieczerowanie, 77
 - zapis wrostkowy, 46
 - zapytanie
 - kategorie, 146
 - kontynuacja, 154
 - LINQ, 86, *Patrz:* LINQ
 - z wieloma generatorami, 155
 - zintegrowane, *Patrz:* LINQ
 - zasada
 - przypisań oznaczonych, 39
 - skróconego obliczania, 30
 - zdarzenie, 8, 47, 62, 87, 95, 108, 168
 - abstrakcyjne, 109
 - akcesor, 112
 - jawny, 112
 - EventArgs, 107
 - odpalanie, 110
 - przesłaniane, 109
 - schemat standardowy, 109
 - statyczne, 109
 - wirtualne, 75, 109
 - z pustym kodem obsługi zdarzenia, 117
 - zapieczerowane, 109
 - zestaw, 9
 - zaprzyjaźniony, 86
 - zmienna, 38, 46
 - deklaracja, 45, 51
 - delegatu, 102
 - inicjalizacja, 45
 - lokalna, 7, 38, 40, 52
 - typowana niejawnie, 46
 - typu referencyjnego, 19
 - typu wartościowego, 18
 - wciągnięta, 114, 115
 - zewnętrza, 114
 - znacznik XML, 13
 - znak, 31
 - !, 30, 49, 132, 134
 - !=, 29, 30, 50, 84, 92, 132, 134, 135
 - %, 25, 49, 132, 134
 - &, 26, 30, 49, 50, 91, 92, 132, 133, 134, 185, 187
 - &&, 30, 50
 - &=, 51
 - *, 49, 132, 134, 185, 187, *Patrz:* operator mnożenia
 - *=, 51
 - ., 12, 48, 59

znak

/, 13, 49, 132, 134
/=, 51, 134
;, 12
?;, 50
??, 133
@, 11, 32
\, 31, 32
^, 26, 50, 92, 132, 134
^=, 51
{, 12
|, 26, 30, 50, 91, 92, 132, 133, 134
| |, 30, 50
| =, 51
}, 12
~, 26, 49, 92, 132, 134
+, 33, 49, 50, 92, 103, 104, 132, 134
++, 25, 26, 48, 49, 92, 132, 134
+=, 51, 92, 103, 104, 134
<, 30, 33, 50, 92, 132, 134, 135

<<, 26, 132, 134
<<=, 51
<=, 30, 50, 92, 132, 135
=, 13, 47, 51, 92
-=, 51, 92, 103, 104
==, 13, 29, 30, 50, 83, 84, 92, 132,
134, 135
=>, 51
>, 30, 33, 50, 92, 132, 134, 135
>=, 48, 185, 187
>=, 30, 50, 92, 132, 135
>>, 26, 132, 134
>>=, 51
ciąg, 31, 34, 57
 łączenie, 33
 porównywanie, 33
 wyszukiwanie, 33
interpunkcyjny, 12
specjalny, 31, 32
sterujący, 31, 32

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>

GRUPA WYDAWNICZA

 **Helion SA**

C# 5.0



C# to obiektowy język programowania przeznaczony do tworzenia rozwiązań dla platformy .NET. Dzięki licznym zaletom zdobył ogromną popularność wśród programistów i jest jednym z wiodących języków programowania. W jego kolejnej wersji, 5.0, wprowadzono usprawnienia, dzięki którym życie programistów stało się łatwiejsze. Ten podręczny leksykon pozwoli Ci błyskawicznie poznać język C# oraz nowości wprowadzone w wersji 5.

W trakcie lektury poznasz podstawy języka C#, jego składnię, sposób deklarowania zmiennych i tworzenia funkcji. Nauczysz się operować na typach liczbowych, tworzyć pętle, instrukcje warunkowe oraz przestrzenie nazw. Ponadto wkroczysz w świat programowania obiektowego, zaznajamiając się z dziedziczeniem, polimorfizmem i interfejsami. Kolejne strony to coraz bardziej zaawansowana wiedza na temat języka LINQ, wiązań dynamicznych (nowość w C# 5.0) i funkcji asynchronicznych. Ten leksykon to pozycja obowiązkowa dla każdego programisty. Sprawdzi się również w rękach początkujących programistów jako błyskawiczny przewodnik po konstrukcjach języka C#. Warto mieć tę książkę!

Poznaj:

- nowości w C# 5.0
- możliwości LINQ
- zasady programowania obiektowego

Bądź na bieżąco z nowymi możliwościami C# 5.0!

helion.pl
księgarnia
internetowa

Nr katalogowy: 15721



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

🔗 <http://helion.pl/promocje>

Książki najchętniej czytane:

🔗 <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

🔗 <http://helion.pl/nowosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po WIECEJ



KOD KORZYŚCI

ISBN 978-83-246-6273-9



Cena 32,90 zł