

MARCIN LIS

WYDANIE III

C#



PRAKTYCZNY  
KURS

Helion 

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.

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.

Opieka redakcyjna: Ewelina Burska

Projekt okładki: Studio Gravite/Olsztyn  
Obarek, Pokoński, Pazdrijowski, Zaprucki

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto: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/cshpk3>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z ćwiczeniami i listingami wykorzystanymi w książce można znaleźć pod adresem:  
<ftp://ftp.helion.pl/przyklady/cshpk3.zip>

ISBN: 978-83-283-1456-6

Copyright © Helion 2016

Printed in Poland.

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

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

# Spis treści

<b>Wstęp</b> .....	<b>9</b>
Czym jest C#? .....	9
Dla kogo jest ta książka? .....	9
Standardy C# .....	10
<b>Rozdział 1. Zanim zaczniesz programować</b> .....	<b>11</b>
Lekcja 1. Podstawowe koncepcje C# i .NET .....	11
Jak to działa? .....	11
Narzędzia .....	12
Instalacja narzędzi .....	13
Lekcja 2. Pierwsza aplikacja, kompilacja i uruchomienie programu .....	16
.NET Framework .....	16
Visual Studio .....	19
Mono .....	22
MonoDevelop (Xamarin Studio) .....	23
Struktura kodu .....	26
Lekcja 3. Komentarze .....	27
Komentarz blokowy .....	27
Komentarz liniowy .....	28
Komentarz XML .....	29
Ćwiczenia do samodzielnego wykonania .....	30
<b>Rozdział 2. Elementy języka</b> .....	<b>31</b>
Typy danych .....	31
Lekcja 4. Typy danych w C# .....	32
Typy danych w C# .....	32
Zapis wartości (literały) .....	36
Zmienne .....	39
Lekcja 5. Deklaracje i przypisania .....	39
Proste deklaracje .....	39
Deklaracje wielu zmiennych .....	41
Nazwy zmiennych .....	42
Zmienne typów odnośnikowych .....	42
Ćwiczenia do samodzielnego wykonania .....	43

Lekcja 6. Wyprowadzanie danych na ekran .....	43
Wyświetlanie wartości zmiennych .....	43
Wyświetlanie znaków specjalnych .....	46
Instrukcja Console.WriteLine .....	48
Ćwiczenia do samodzielnego wykonania .....	49
Lekcja 7. Operacje na zmiennych .....	49
Operacje arytmetyczne .....	50
Operacje bitowe .....	57
Operacje logiczne .....	61
Operatory przypisania .....	63
Operatory porównywania (relacyjne) .....	64
Pozostałe operatory .....	65
Priorytety operatorów .....	65
Ćwiczenia do samodzielnego wykonania .....	66
Instrukcje sterujące .....	67
Lekcja 8. Instrukcja warunkowa if..else .....	67
Podstawowa postać instrukcji if..else .....	67
Zagnieżdżanie instrukcji if..else .....	69
Instrukcja if..else if .....	72
Ćwiczenia do samodzielnego wykonania .....	75
Lekcja 9. Instrukcja switch i operator warunkowy .....	76
Instrukcja switch .....	76
Przerywanie instrukcji switch .....	79
Operator warunkowy .....	81
Ćwiczenia do samodzielnego wykonania .....	82
Lekcja 10. Pętle .....	82
Pętla for .....	83
Pętla while .....	86
Pętla do..while .....	88
Pętla foreach .....	89
Ćwiczenia do samodzielnego wykonania .....	90
Lekcja 11. Instrukcje break i continue .....	91
Instrukcja break .....	91
Instrukcja continue .....	95
Ćwiczenia do samodzielnego wykonania .....	96
Tablice .....	97
Lekcja 12. Podstawowe operacje na tablicach .....	98
Tworzenie tablic .....	98
Inicjalizacja tablic .....	101
Właściwość Length .....	102
Ćwiczenia do samodzielnego wykonania .....	103
Lekcja 13. Tablice wielowymiarowe .....	104
Tablice dwuwymiarowe .....	104
Tablice tablic .....	107
Tablice dwuwymiarowe i właściwość Length .....	109
Tablice nieregularne .....	111
Ćwiczenia do samodzielnego wykonania .....	115
<b>Rozdział 3. Programowanie obiektowe .....</b>	<b>117</b>
Podstawy .....	117
Lekcja 14. Klasy i obiekty .....	118
Podstawy obiektowości .....	118
Pierwsza klasa .....	119
Jak użyć klasy? .....	121

Metody klas .....	122
Jednostki kompilacji, przestrzenie nazw i zestawy .....	126
Ćwiczenia do samodzielnego wykonania .....	130
Lekcja 15. Argumenty i przeciążanie metod .....	131
Argumenty metod .....	131
Obiekt jako argument .....	134
Przeciążanie metod .....	138
Argumenty metody Main .....	139
Sposoby przekazywania argumentów .....	140
Definicje metod za pomocą wyrażeń lambda .....	143
Ćwiczenia do samodzielnego wykonania .....	144
Lekcja 16. Konstruktory i destruktory .....	145
Czym jest konstruktor? .....	145
Argumenty konstruktorów .....	148
Przeciążanie konstruktorów .....	149
Słowo kluczowe this .....	151
Niszczenie obiektu .....	154
Ćwiczenia do samodzielnego wykonania .....	155
Dziedziczenie .....	156
Lekcja 17. Klasy potomne .....	156
Dziedziczenie .....	156
Konstruktory klasy bazowej i potomnej .....	160
Ćwiczenia do samodzielnego wykonania .....	164
Lekcja 18. Modyfikatory dostępu .....	164
Określanie reguł dostępu .....	165
Dlaczego ukrywamy wewnątrz klasy? .....	170
Jak zabronić dziedziczenia? .....	174
Tylko do odczytu .....	175
Ćwiczenia do samodzielnego wykonania .....	178
Lekcja 19. Przesłanie metod i składowe statyczne .....	179
Przesłanie metod .....	179
Przesłanie pól .....	182
Składowe statyczne .....	183
Ćwiczenia do samodzielnego wykonania .....	186
Lekcja 20. Właściwości i struktury .....	186
Właściwości .....	187
Struktury .....	196
Ćwiczenia do samodzielnego wykonania .....	200

## **Rozdział 4. Wyjątki i obsługa błędów ..... 203**

Lekcja 21. Blok try...catch .....	203
Badanie poprawności danych .....	203
Wyjątki w C# .....	207
Ćwiczenia do samodzielnego wykonania .....	211
Lekcja 22. Wyjątki to obiekty .....	212
Dzielenie przez zero .....	212
Wyjątek jest obiektem .....	213
Hierarchia wyjątków .....	214
Przechwytywanie wielu wyjątków .....	215
Zagnieżdżanie bloków try...catch .....	218
Ćwiczenia do samodzielnego wykonania .....	220

Lekcja 23. Własne wyjątki .....	220
Zgłaszanie wyjątków .....	221
Ponowne zgłoszenie przechwyconego wyjątku .....	223
Tworzenie własnych wyjątków .....	225
Wyjątki warunkowe .....	226
Sekcja finally .....	228
Ćwiczenia do samodzielnego wykonania .....	231
<b>Rozdział 5. System wejścia-wyjścia .....</b>	<b>233</b>
Lekcja 24. Ciągi znaków .....	233
Znaki i łańcuchy znakowe .....	233
Znaki specjalne .....	237
Zamiana ciągów na wartości .....	238
Formatowanie danych .....	240
Przetwarzanie ciągów .....	242
Ćwiczenia do samodzielnego wykonania .....	247
Lekcja 25. Standardowe wejście i wyjście .....	247
Klasa Console i odczyt znaków .....	248
Wczytywanie tekstu z klawiatury .....	255
Wprowadzanie liczb .....	256
Ćwiczenia do samodzielnego wykonania .....	257
Lekcja 26. Operacje na systemie plików .....	258
Klasa FileSystemInfo .....	258
Operacje na katalogach .....	259
Operacje na plikach .....	266
Ćwiczenia do samodzielnego wykonania .....	271
Lekcja 27. Zapis i odczyt plików .....	271
Klasa FileStream .....	272
Podstawowe operacje odczytu i zapisu .....	274
Operacje strumieniowe .....	278
Ćwiczenia do samodzielnego wykonania .....	287
<b>Rozdział 6. Zaawansowane zagadnienia programowania obiektowego .....</b>	<b>289</b>
Polimorfizm .....	289
Lekcja 28. Konwersje typów i rzutowanie obiektów .....	289
Konwersje typów prostych .....	290
Rzutowanie typów obiektowych .....	291
Rzutowanie na typ Object .....	295
Typy proste też są obiektowe! .....	297
Ćwiczenia do samodzielnego wykonania .....	299
Lekcja 29. Późne wiązanie i wywoływanie metod klas pochodnych .....	299
Rzeczywisty typ obiektu .....	300
Dziedziczenie a wywoływanie metod .....	302
Dziedziczenie a metody prywatne .....	307
Ćwiczenia do samodzielnego wykonania .....	308
Lekcja 30. Konstruktory oraz klasy abstrakcyjne .....	309
Klasy i metody abstrakcyjne .....	309
Wywołania konstruktorów .....	313
Wywoływanie metod w konstruktorach .....	316
Ćwiczenia do samodzielnego wykonania .....	318
Interfejsy .....	319
Lekcja 31. Tworzenie interfejsów .....	319
Czym są interfejsy? .....	319
Interfejsy a hierarchia klas .....	322

Interfejsy i właściwości .....	324
Ćwiczenia do samodzielnego wykonania .....	326
Lekcja 32. Implementacja kilku interfejsów .....	326
Implementowanie wielu interfejsów .....	327
Konflikty nazw .....	328
Dziedziczenie interfejsów .....	331
Ćwiczenia do samodzielnego wykonania .....	333
Klasy zagnieżdżone .....	334
Lekcja 33. Klasa wewnątrz klasy .....	334
Tworzenie klas zagnieżdżonych .....	334
Kilka klas zagnieżdżonych .....	336
Składowe klasy zagnieżdżonych .....	338
Obiekty klas zagnieżdżonych .....	339
Rodzaje klas wewnętrznych .....	342
Dostęp do składowych klasy zewnętrznej .....	344
Ćwiczenia do samodzielnego wykonania .....	345
Typy uogólnione .....	346
Lekcja 34. Kontrola typów i typy uogólnione .....	346
Jak zbudować kontener? .....	346
Przechowywanie dowolnych danych .....	350
Problem kontroli typów .....	352
Korzystanie z typów uogólnionych .....	353
Ćwiczenia do samodzielnego wykonania .....	356
<b>Rozdział 7. Aplikacje z interfejsem graficznym .....</b>	<b>359</b>
Lekcja 35. Tworzenie okien .....	359
Pierwsze okno .....	359
Klasa Form .....	361
Tworzenie menu .....	366
Ćwiczenia do samodzielnego wykonania .....	370
Lekcja 36. Delegacje i zdarzenia .....	371
Koncepcja zdarzeń i delegacji .....	371
Tworzenie delegacji .....	371
Delegacja jako funkcja zwrotna .....	375
Delegacja powiązana z wieloma metodami .....	379
Zdarzenia .....	381
Ćwiczenia do samodzielnego wykonania .....	391
Lekcja 37. Komponenty graficzne .....	392
Wyświetlanie komunikatów .....	392
Obsługa zdarzeń .....	393
Menu .....	395
Etykiety .....	397
Przyciski .....	399
Pola tekstowe .....	401
Listy rozwijane .....	404
Ćwiczenia do samodzielnego wykonania .....	407
<b>Zakończenie .....</b>	<b>409</b>
<b>Skorowidz .....</b>	<b>410</b>





## Rozdział 3.

# Programowanie obiektowe

Każdy program w C# składa się z jednej lub wielu klas. W dotychczas prezentowanych przykładach była to tylko jednak klasa o nazwie Program. Przypomnijmy sobie naszą pierwszą aplikację, wyświetlającą na ekranie napis. Jej kod wyglądał następująco:

```
using System;

public class Program
{
    public static void Main()
    {
        Console.WriteLine("Mój pierwszy program!");
    }
}
```

Założyliśmy wtedy, że szkielet kolejnych programów, na których demonstrowano struktury języka programowania, ma właśnie tak wyglądać. Teraz nadszedł czas, aby dowiedzieć się, dlaczego właśnie tak. Wszystko zostanie wyjaśnione w niniejszym rozdziale.

## Podstawy

Pierwsza część rozdziału 3. składa się z trzech lekcji, w których podjęto tematykę podstaw programowania obiektowego w C#. W lekcji 14. jest omawiana budowa klas oraz tworzenie obiektów. Zostały w niej przedstawione pola i metody, sposoby ich deklaracji oraz wywoływania. Lekcja 15. jest poświęcona argumentom metod oraz technice przeciążania metod, została w niej również przybliżona wykorzystywana już wcześniej metoda `Main`. W ostatniej, 16. lekcji, zaprezentowano temat konstruktorów, czyli specjalnych metod wywoływanych podczas tworzenia obiektów.

## Lekcja 14. Klasy i obiekty

Lekcja 14. rozpoczyna rozdział przedstawiający podstawy programowania obiektowego w C#. Najważniejsze pojęcia zostaną tu wyjaśnione na praktycznych przykładach. Zajmiemy się tworzeniem klas, ich strukturą i deklaracjami, przeanalizujemy związek między klasą i obiektem. Zostaną przedstawione składowe klasy, czyli pola i metody, będzie też wyjaśnione, czym są wartości domyślne pól. Opisane zostaną również relacje między zadeklarowaną na stosie zmienną obiektową (inaczej referencyjną, odnośnikową) a utworzonym na stercie obiektem.

### Podstawy obiektowości

Program w C# składa się z klas, które są z kolei opisami obiektów. To podstawowe pojęcia związane z programowaniem obiektowym. Osoby, które nie zetknęły się dotychczas z programowaniem obiektowym, mogą potraktować **obiekt** (ang. *object*) jako pewien byt programistyczny, który może przechowywać dane i wykonywać operacje, czyli różne zadania. **Klasa** (ang. *class*) to z kolei definicja, opis takiego obiektu.

Skoro klasa definiuje obiekt, jest zatem również jego typem. Czym jest **typ obiektu**? Przytoczmy jedną z definicji: „Typ jest przypisany zmiennej, wyrażeniu lub innemu bytowi programistycznemu (danej, obiektowi, funkcji, procedurze, operacji, metodzie, parametrowi, modułowi, wyjątkowi, zdarzeniu). Specyfikuje on rodzaj wartości, które może przybierać ten byt. (...) Jest to również ograniczenie kontekstu, w którym odwołanie do tego bytu może być użyte w programie”<sup>1</sup>. Innymi słowy, typ obiektu określa po prostu, czym jest dany obiekt. Tak samo jak miało to miejsce w przypadku zmiennych typów prostych. Jeśli mieliśmy zmienną typu `int`, to mogła ona przechowywać wartości całkowite. Z obiektami jest podobnie. Zmienna obiektowa hipotetycznej klasy `Punkt` może przechowywać obiekty klasy (typu) `Punkt`<sup>2</sup>. Klasa to zatem nic innego jak definicja nowego typu danych.

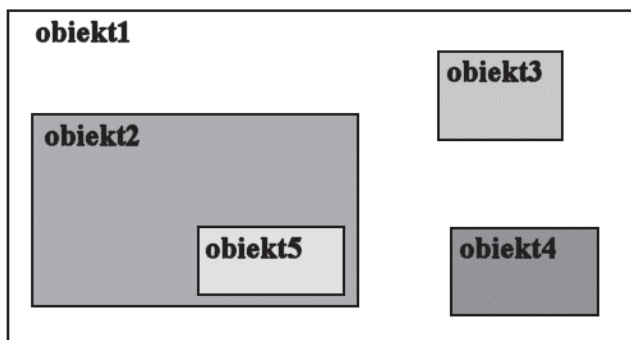
Co może być obiektem? Tak naprawdę — wszystko. W życiu codziennym mianem tym określić możemy stół, krzesło, komputer, dom, samochód, radio... Każdy z obiektów ma pewne cechy, właściwości, które go opisują: wielkość, kolor, powierzchnię, wysokość. Co więcej, każdy obiekt może składać się z innych obiektów (rysunek 3.1). Na przykład mieszkanie składa się z poszczególnych pomieszczeń, z których każde może być obiektem; w każdym pomieszczeniu mamy z kolei inne obiekty: sprzęty domowe, meble itd.

Obiekty oprócz tego, że mają właściwości, mogą wykonywać różne funkcje, zadania. Innymi słowy, każdy obiekt ma przypisany pewien zestaw poleceń, które potrafi wykonywać. Na przykład samochód „rozumie” polecenia „uruchom silnik”, „wyłącz silnik”, „skreć w prawo”, „przyspiesz” itp. Funkcje te składają się na pewien interfejs udostępniany nam przez tenże samochód. Dzięki interfejsowi możemy wpływać na zachowanie samochodu i wydawać mu polecenia.

<sup>1</sup> K. Subieta, *Wytwarzanie, integracja i testowanie systemów informatycznych*, PJWSTK, Warszawa 1997.

<sup>2</sup> W dalszej części książki zostanie pokazane, że takiej zmiennej można również przypisać obiekty klas potomnych lub nadrzędnych w stosunku do klasy `Punkt`.

**Rysunek 3.1.**  
Obiekt może zawierać inne obiekty



W programowaniu jest bardzo podobnie. Za pomocą klas staramy się opisać obiekty, ich właściwości, zbudować konstrukcje, interfejsy, dzięki którym będziemy mogli wydawać polecenia realizowane potem przez obiekty. Obiekt powstaje jednak dopiero w trakcie działania programu jako instancja (wystąpienie, egzemplarz) danej klasy. Obiektów danej klasy może być bardzo dużo. Jeśli na przykład klasą będzie *Samochód*, to instancją tej klasy będzie konkretny egzemplarz o danym numerze seryjnym.

Ponieważ dla osób nieobeznanych z programowaniem obiektowym może to wszystko brzmieć nieco zawile, od razu zobaczymy, jak to będzie wyglądało w praktyce.

## Pierwsza klasa

Załóżmy, że pisany przez nas program wymaga przechowywania danych odnoszących się do punktów na płaszczyźnie, ekranie. Każdy taki punkt jest charakteryzowany przez dwie wartości: współrzędną  $x$  oraz współrzędną  $y$ . Utwórzmy więc klasę opisującą obiekty tego typu. Schematyczny szkielet klasy wygląda następująco:

```
class nazwa_klasy
{
    //treść klasy
}
```

W treści klasy definiujemy pola i metody. Pola służą do przechowywania danych, metody do wykonywania różnych operacji. W przypadku klasy, która ma przechowywać dane dotyczące współrzędnych  $x$  i  $y$ , wystarczą dwa pola typu `int` (przy założeniu, że wystarczające będzie przechowywanie wyłącznie współrzędnych całkowitych). Pozostaje jeszcze wybór nazwy dla takiej klasy. Występują tu takie same ograniczenia jak w przypadku nazewnictwa zmiennych (por. lekcja 5.), czyli nazwa klasy może składać się jedynie z liter (zarówno małych, jak i dużych), cyfr oraz znaku podkreślenia, ale nie może zaczynać się od cyfry. Można stosować znaki polskie znaki (choć wielu programistów używa wyłącznie alfabetu łacińskiego, nawet jeśli nazwy pochodzą z języka polskiego). Przyjęte jest również, że w nazwach nie używa się znaku podkreślenia.

Naszą klasę nazwiemy zatem, jakżeby inaczej, `Punkt` i będzie ona miała postać widoczną na listingu 3.1. Kod ten zapiszemy w pliku o nazwie *Punkt.cs*.

**Listing 3.1.** Klasa przechowująca współrzędne punktów

---

```
class Punkt
{
    int x;
    int y;
}
```

---

Ta klasa zawiera dwa pola o nazwach *x* i *y*, które opisują współrzędne położenia punktu. Pola definiujemy w taki sam sposób jak zmienne.

Kiedy mamy zdefiniowaną klasę *Punkt*, możemy zadeklarować zmienną typu *Punkt*. Robimy to podobnie jak wtedy, gdy deklarowaliśmy zmienne typów prostych (np. *short*, *int*, *char*), czyli pisząc:

```
typ_zmiennej nazwa_zmiennej;
```

Ponieważ typem zmiennej jest nazwa klasy (klasa to definicja typu danych), to jeśli nazwą zmiennej ma być *przykładowyPunkt*, deklaracja przyjmie postać:

```
Punkt przykładowyPunkt;
```

W ten sposób powstała zmienna odnośnikowa (referencyjna, obiektowa), która domyślnie jest pusta, tzn. nie zawiera żadnych danych. Dokładniej rzecz ujmując, po deklaracji zmienna taka zawiera wartość specjalną *null*, która określa, że nie ma ona odniesienia do żadnego obiektu. Musimy więc sami utworzyć obiekt klasy *Punkt* i przypisać go tej zmiennej<sup>3</sup>. Obiekty tworzy się za pomocą operatora *new* w postaci:

```
new nazwa_klasy();
```

zatem cała konstrukcja schematycznie wyglądać będzie następująco:

```
nazwa_klasy nazwa_zmiennej = new nazwa_klasy();
```

a w przypadku naszej klasy *Punkt*:

```
Punkt przykładowyPunkt = new Punkt();
```

Oczywiście, podobnie jak w przypadku zmiennych typów prostych (por. lekcja 5.), również i tutaj można oddzielić deklarację zmiennej od jej inicjalizacji, zatem również poprawna jest konstrukcja w postaci:

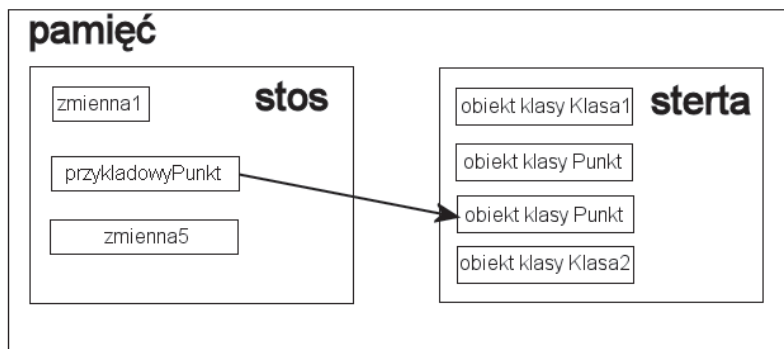
```
Punkt przykładowyPunkt;
przykładowyPunkt = new Punkt();
```

Konieczne trzeba sobie dobrze uzmysłowić, że po wykonaniu tych instrukcji w pamięci powstają dwie różne struktury. Pierwszą z nich jest powstała na tak zwanym *stosie* (ang. *stack*) zmienna referencyjna *przykładowyPunkt*, drugą jest powstały na tak zwanej *stercie* (ang. *heap*) obiekt klasy (typu) *Punkt*. Zmienna *przykładowyPunkt* zawiera odniesienie do przypisanego jej obiektu klasy *Punkt* i tylko poprzez nią możemy się do tego obiektu odwoływać. Schematycznie zobrazowano to na rysunku 3.2.

---

<sup>3</sup> Osoby programujące w C++ powinny zwrócić na to uwagę, gdyż w tym języku już sama deklaracja zmiennej typu klasowego powoduje wywołanie domyślnego konstruktora i utworzenie obiektu.

**Rysunek 3.2.**  
Zależność  
między zmienną  
odnośnikową  
a wskazywanym  
przez nią obiektem



Jeśli chcemy odwołać się do danego pola klasy, korzystamy z operatora `.` (kropka), czyli używamy konstrukcji:

```
nazwa_zmiennej_obiektowej.nazwa_pola_obiektu
```

Przykładowo przypisanie wartości 100 polu `x` obiektu klasy `Punkt` reprezentowanego przez zmienną `przykładowyPunkt` będzie wyglądało następująco:

```
przykładowyPunkt.x = 100;
```

## Jak użyć klasy?

Spróbujmy teraz przekonać się, że obiekt klasy `Punkt` faktycznie jest w stanie przechowywać dane. Jak wiadomo z poprzednich rozdziałów, aby program mógł zostać uruchomiony, musi zawierać metodę `Main` (więcej o metodach już w kolejnym podpunkcie, a o metodzie `Main` w jednej z kolejnych lekcji). Dopiszmy więc do klasy `Punkt` taką metodę, która utworzy obiekt, przypisze jego polom pewne wartości oraz wyświetli je na ekranie. Kod programu realizującego takie zadanie jest widoczny na listingu 3.2.

**Listing 3.2.** *Użycie klasy Punkt*

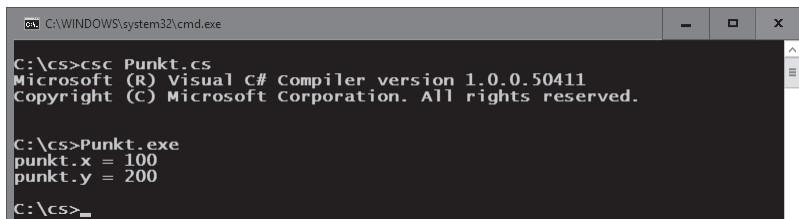
```
using System;

class Punkt
{
    int x;
    int y;

    public static void Main()
    {
        Punkt punkt1 = new Punkt();
        punkt1.x = 100;
        punkt1.y = 200;
        Console.WriteLine("punkt.x = " + punkt1.x);
        Console.WriteLine("punkt.y = " + punkt1.y);
    }
}
```

Struktura klasy Punkt jest taka sama jak w przypadku listingu 3.1, z tą różnicą, że do jej treści została dodana metoda Main. W tej metodzie deklarujemy zmienną klasy Punkt o nazwie punkt1 i przypisujemy jej nowo utworzony obiekt tej klasy. Dokonujemy zatem jednocześnie deklaracji i inicjalizacji. Od tej chwili zmienna punkt1 wskazuje na obiekt klasy Punkt, możemy się zatem posługiwać nią tak, jakbyśmy posługiwali się samym obiektem. Pisząc zatem punkt1.x = 100, przypisujemy wartość 100 polu x, a pisząc punkt.y = 200, przypisujemy wartość 200 polu y. W ostatnich dwóch liniach korzystamy z instrukcji Console.WriteLine, aby wyświetlić wartość obu pól na ekranie. Efekt jest widoczny na rysunku 3.3.

**Rysunek 3.3.**  
Wynik działania  
klasy Punkt  
z listingu 3.2



```
C:\WINDOWS\system32\cmd.exe

C:\>csc Punkt.cs
Microsoft (R) Visual C# Compiler version 1.0.0.50411
Copyright (C) Microsoft Corporation. All rights reserved.

C:\>Punkt.exe
punkt.x = 100
punkt.y = 200

C:\>_
```

## Metody klas

Klasy oprócz pól przechowujących dane zawierają także metody, które wykonują zapisane przez programistę operacje. Definiujemy je w ciele (czyli wewnątrz) klasy pomiędzy znakami nawiasu klamrowego. Każda metoda może przyjmować argumenty oraz zwracać wynik. Schematyczna deklaracja metody wygląda następująco:

```
typ_wyniku nazwa_metody(argumenty_metody)
{
    instrukcje metody
}
```

Po umieszczeniu w ciele klasy deklaracja taka będzie natomiast wyglądała tak:

```
class nazwa_klasy
{
    typ_wyniku nazwa_metody(argumenty_metody)
    {
        instrukcje metody
    }
}
```

Jeśli metoda nie zwraca żadnego wyniku, jako typ wyniku należy zastosować słowo void; jeśli natomiast nie przyjmuje żadnych parametrów, pomiędzy znakami nawiasu okrągłego nie należy nic wpisywać. Aby zobaczyć, jak to wygląda w praktyce, do klasy Punkt dodamy prostą metodę, której zadaniem będzie wyświetlenie wartości współrzędnych x i y na ekranie. Nadamy jej nazwę WyświetlWspolrzedne, zatem jej wygląd będzie następujący:

```
void WyświetlWspolrzedne()
{
    Console.WriteLine("współrzędna x = " + x);
    Console.WriteLine("współrzędna y = " + y);
}
```

Słowo `void` oznacza, że metoda nie zwraca żadnego wyniku, a brak argumentów pomiędzy znakami nawiasu okrągłego wskazuje, że metoda ta żadnych argumentów nie przyjmuje. We wnętrzu metody znajdują się dwie dobrze nam znane instrukcje, które wyświetlają na ekranie współrzędne punktu. Po umieszczeniu powyższego kodu w klasie `Punkt` przyjmie ona postać widoczną na listingu 3.3.

---

**Listing 3.3.** *Dodanie metody do klasy `Punkt`*

---

```
using System;

class Punkt
{
    int x;
    int y;

    void WyświetlWspolrzedne()
    {
        Console.WriteLine("współrzędna x = " + x);
        Console.WriteLine("współrzędna y = " + y);
    }
}
```

---

Po utworzeniu obiektu danej klasy możemy **wywołać** (uruchomić) metodę w taki sam sposób, w jaki odwołujemy się do pól klasy, tzn. korzystając z operatora `.` (kropka). Jeśli zatem przykładowa zmienna `punkt1` zawiera referencję do obiektu klasy `Punkt`, prawidłowym wywołaniem metody `WyświetlWspolrzedne` będzie:

```
punkt1.WyświetlWspolrzedne();
```

Ogólnie wywołanie metody wygląda następująco:

```
nazwa_zmiennej.nazwa_metody(argumenty_metody);
```

Oczywiście, jeśli dana metoda nie ma argumentów, po prostu je pomijamy. Przy czym termin *wywołanie* oznacza po prostu wykonanie kodu (instrukcji) zawartego w metodzie.

Użyjmy zatem metody `Main` do przetestowania nowej konstrukcji. W tym celu zmodyfikujemy program z listingu 3.2 tak, aby wykorzystywał metodę `WyświetlWspolrzedne`. Odpowiedni kod jest zaprezentowany na listingu 3.4. Wynik jego działania jest łatwy do przewidzenia (rysunek 3.4).

---

**Listing 3.4.** *Wywołanie metody `WyświetlWspolrzedne`*

---

```
using System;

class Punkt
{
    int x;
    int y;

    void WyświetlWspolrzedne()
    {
        Console.WriteLine("współrzędna x = " + x);
        Console.WriteLine("współrzędna y = " + y);
    }
}
```

```

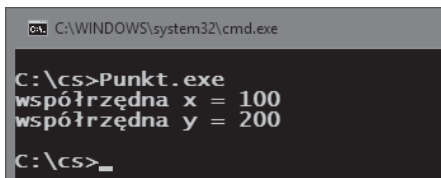
    }

    public static void Main()
    {
        Punkt punkt1 = new Punkt();
        punkt1.x = 100;
        punkt1.y = 200;
        punkt1.WyswietlWspolrzedne();
    }
}

```

### Rysunek 3.4.

Wynik działania metody  
WyswietlWspolrzedne  
z klasy Punkt



```

C:\WINDOWS\system32\cmd.exe

C:\cs>Punkt.exe
współrzędna x = 100
współrzędna y = 200

C:\cs>_

```

Przedstawiony kod jest w istocie złożeniem przykładów z listingów 3.2 i 3.3. Klasa Punkt z listingu 3.3 została uzupełniona o nieco zmodyfikowany kod metody Main, pobrany z listingu 3.2. W metodzie tej jest więc tworzony nowy obiekt typu Punkt i ustalone są wartości jego pól x i y. Do wyświetlenia wartości zapisanych w x i y jest natomiast używana metoda WyswietlWspolrzedne.

Zobaczmy teraz, w jaki sposób napisać metody, które będą mogły zwracać wyniki. Typ wyniku należy podać przed nazwą metody, zatem jeśli ma ona zwracać wartość typu int, deklaracja powinna wyglądać następująco:

```

int nazwa_metody()
{
    //instrukcje metody
}

```

Sam wynik zwracamy natomiast przez zastosowanie instrukcji return. Najlepiej zobaczyć to na praktycznym przykładzie. Do klasy Punkt dodamy zatem dwie metody — jedna będzie podawała wartość współrzędnej x, druga y. Nazwiemy je odpowiednio PobierzX i PobierzY. Wygląd metody PobierzX będzie następujący:

```

int PobierzX()
{
    return x;
}

```

Przed nazwą metody znajduje się określenie typu zwracanego przez nią wyniku — skoro jest to int, oznacza to, że metoda ta musi zwrócić jako wynik liczbę całkowitą z przedziału określonego przez typ int (tabela 2.1). Wynik jest zwracany dzięki instrukcji return. Zapis return x oznacza zwrócenie przez metodę wartości zapisanej w polu x.

Jak łatwo się domyślić, metoda PobierzY będzie wyglądała analogicznie, z tym że będzie w niej zwracana wartość zapisana w polu y. Pełny kod klasy Punkt po dodaniu tych dwóch metod będzie wyglądał tak, jak przedstawiono na listingu 3.5.



**Listing 3.5.** *Metody zwracające wyniki*

```
using System;

class Punkt
{
    int x;
    int y;

    int PobierzX()
    {
        return x;
    }

    int PobierzY()
    {
        return y;
    }

    void WyświetlWspolrzedne()
    {
        Console.WriteLine("współrzędna x = " + x);
        Console.WriteLine("współrzędna y = " + y);
    }
}
```

Jeśli teraz zechcemy przekonać się, jak działają nowe metody, możemy wyposażyc klasę Punkt w metodę Main testującą ich działanie. Mogłaby ona mieć postać widoczną na listingu 3.6.

**Listing 3.6.** *Metoda Main testująca działanie klasy Punkt*

```
public static void Main()
{
    Punkt punkt1 = new Punkt();
    punkt1.x = 100;
    punkt1.y = 200;
    int wspX = punkt1.PobierzX();
    int wspY = punkt1.PobierzY();
    Console.WriteLine("współrzędna x = " + wspX);
    Console.WriteLine("współrzędna y = " + wspY);
}
```

Początek kodu jest tu taki sam jak we wcześniej prezentowanych przykładach — powstaje obiekt typu Punkt i są w nim zapisywane przykładowe współrzędne. Następnie tworzone są dwie zmienne typu int: `wspX` i `wspY`. Pierwszej przypisywana jest wartość zwrócona przez metodę `PobierzX`, a drugiej — wartość zwrócona przez metodę `PobierzY`. Wartości zapisane w zmiennych są następnie wyświetlane w standardowy sposób na ekranie.

Warto tu zauważyć, że zmienne `wspX` i `wspY` pełnią funkcję pomocniczą — dzięki nim kod jest czytelniejszy. Nic jednak nie stoi na przeszkodzie, aby wartości zwrócone przez metody były używane bezpośrednio w instrukcjach `Console.WriteLine`<sup>4</sup>. Metoda

<sup>4</sup> Po wyjaśnieniach przedstawionych w lekcji 3. można się domyślić, że to, co do tej pory było nazywane instrukcją `WriteLine`, jest w rzeczywistości wywołaniem metody o nazwie `WriteLine`.

Main mogłaby więc mieć również postać przedstawioną na listingu 3.7. Efekt działania byłby taki sam.

**Listing 3.7.** Alternatywna wersja metody Main

```
public static void Main()
{
    Punkt punkt1 = new Punkt();
    punkt1.x = 100;
    punkt1.y = 200;
    Console.WriteLine("współrzędna x = " + punkt1.PobierzX());
    Console.WriteLine("współrzędna y = " + punkt1.PobierzY());
}
```

## Jednostki kompilacji, przestrzenie nazw i zestawy

Każdą klasę można zapisać w pliku o dowolnej nazwie. Często przyjmuje się jednak, że nazwa pliku powinna być zgodna z nazwą klasy. Jeśli zatem istnieje klasa Punkt, to jej kod powinien znaleźć się w pliku *Punkt.cs*. W jednym pliku może się też znaleźć kilka klas. Wówczas jednak zazwyczaj są to tylko jedna klasa główna oraz dodatkowe klasy pomocnicze. W przypadku prostych aplikacji tych zasad nie trzeba przestrzegać, ale w przypadku większych programów umieszczenie całej struktury kodu w jednym pliku spowodowałoby duże trudności w zarządzaniu nim. Pojedynczy plik można nazywać jednostką kompilacji lub modulem.

Wszystkie dotychczasowe przykłady składały się zawsze z jednej klasy zapisywanej w jednym pliku. Zobaczmy więc, jak mogą współpracować ze sobą dwie klasy. Na listingu 3.8 znajduje się nieco zmodyfikowana treść klasy Punkt z listingu 3.1. Przed składowymi zostały dodane słowa `public`, dzięki którym będzie istniała możliwość odwoływania się do nich z innych klas. Ta kwestia zostanie wyjaśniona dokładniej w jednej z kolejnych lekcji. Na listingu 3.9 jest natomiast widoczny kod klasy Program, która korzysta z klasy Punkt. Tak więc treść z listingu 3.8 zapiszemy w pliku o nazwie *Punkt.cs*, a kod z listingu 3.9 w pliku *Program.cs*.

**Listing 3.8.** Prosta klasa Punkt

```
class Punkt
{
    public int x;
    public int y;
}
```

**Listing 3.9.** Klasa Program korzystająca z obiektu klasy Punkt

```
using System;

public class Program
{
    public static void Main()
    {
        Punkt punkt1 = new Punkt();
    }
}
```

```

    punkt1.x = 100;
    punkt1.y = 200;
    Console.WriteLine("punkt1.x = " + punkt1.x);
    Console.WriteLine("punkt1.y = " + punkt1.y);
}
}

```

W klasie Program znajduje się metoda Main, od której rozpoczyna się wykonywanie kodu aplikacji. W tej metodzie tworzony jest obiekt punkt1 klasy Punkt, jego składowym przypisywane są wartości 100 i 200, a następnie są one wyświetlane na ekranie. Tego typu konstrukcje były wykorzystywane już kilkakrotnie we wcześniejszych przykładach.

Jak teraz przetworzyć oba kody na plik wykonywalny? Nie jest to skomplikowane, po prostu nazwy obu plików (Program.cs i Punkt.cs) należy zastosować jako argumenty wywołania kompilatora, czyli w wierszu poleceń wydać komendę:

```
csc Program.cs Punkt.cs
```

Trzeba też wiedzieć, że plik wykonywalny powstały po kompilacji nie zawiera tylko kodu wykonywalnego. W rzeczywistości kod wykonywany na platformie .NET składa się z tak zwanych zestawów (ang. *assembly*). Pojedynczy **zestaw** składa się z manifestu, metadanych oraz kodu języka pośredniego IL. **Manifest** to wszelkie informacje o zestawie, takie jak nazwy plików składowych, odwołania do innych zestawów, numer wersji itp. **Metadane** natomiast to opis danych i kodu języka pośredniego w danym zestawie, zawierający m.in. definicje zastosowanych typów danych.

Wszystko to może być umieszczone w jednym lub też kilku plikach (*exe*, *dll*). We wszystkich przykładach w tej książce będziemy mieli do czynienia tylko z zestawami jednoplikowymi i będą to pliki wykonywalne typu *exe*, generowane automatycznie przez kompilator, tak że nie będziemy musieli zagłębiać się w te kwestie. Nie można jednak pominąć zagadnienia przestrzeni nazw.

Przestrzeń nazw to ograniczenie widoczności danej nazwy, ograniczenie kontekstu, w którym jest ona rozpoznawana. Czemu to służy? Otóż pojedyncza aplikacja może się składać z bardzo dużej liczby klas, a jeszcze więcej klas znajduje się w bibliotekach udostępnianych przez .NET. Co więcej, nad jednym projektem zwykle pracują zespoły programistów. W takiej sytuacji nietrudno o pojawianie się konfliktów nazw, czyli powstawanie klas o takich samych nazwach. Tymczasem nazwa każdej klasy musi być unikatowa. Ten problem rozwiązują właśnie przestrzenie nazw. Jeśli bowiem klasa zostanie umieszczona w danej przestrzeni, to będzie widoczna tylko w niej. Będą więc mogły istnieć klasy o takiej samej nazwie, o ile tylko zostaną umieszczone w różnych przestrzeniach nazw. Taką przestrzeń definiuje się za pomocą słowa *namespace*, a jej składowe należy umieścić w występującym dalej nawiasie klamrowym. Schematycznie wygląda to tak:

```

namespace nazwa_przeznieni
{
    elementy przestrzeni nazw
}

```



# Skorowidz

.NET Framework, 13, 16

## A

- akcesor set, 191
- aliasy, 297
- aplikacje
  - konsolowe, 19
  - okienkowe, 361
- argumenty, 374
  - konstruktorów, 148
  - metody, 131
  - metody Main, 139
- automatyczne
  - konwersje wartości, 54
  - wywołanie konstruktora, 313

## B

- badanie poprawności danych, 203
- bitowa alternatywa wykluczająca, 60
- blok
  - default, 80
  - finally, 229
  - try...catch, 203, 208, 218
- błąd
  - aplikacji, 100
  - kompilacji, 55, 175, 315
- błędna
  - hierarchia wyjątków, 217
  - implementacja interfejsów, 330

## C

- chronione pola, 169
- ciągi znaków, 35, 233, 35, 233
- CIL, Common Intermediate Language, 12
- CLR, Common Language Runtime, 12
- CLS, Common Language Specification, 12

## D

- deklaracja, 39
  - metody, 122
  - zmiennej, 39
  - wielu zmiennych, 41
- dekrementacja, 51
- delegacja, 371, 379
  - OnUjemneEventDelegate, 387
- destruktor, 145, 154
- dodawanie
  - delegacji, 381
  - metody, 123
  - procedur obsługi, 385
  - znaków, 236
- dokument XML, 29
- dostęp
  - chroniony, protected, 169
  - do klasy, 165
  - do obiektu generującego zdarzenie, 386
  - do składowych klasy zagnieżdżonej, 338
  - do składowych klasy zewnętrznej, 344
  - prywatny, private, 168
  - publiczny, public, 166

dynamiczna tablica, 347, 350  
 dyrektywa using, 129  
 dziedziczenie, 156, 174, 302, 307, 322  
   interfejsu, 200, 323, 331  
   struktury, 199  
 dzielenie przez zero, 212

**E**

etykiety, 397

**F**

FCL, Framework Class Library, 12  
 formatowanie danych, 240  
 funkcje zwrotne, 375

**H**

hierarchia klas, 322  
 hierarchia wyjątków, 214

**I**

IDE, Integrated Development Environment, 15  
 iloczyn  
   bitowy, 58  
   logiczny (&&), 61  
   logiczny (&), 62  
 implementacja interfejsów, 325, 330  
 informacja o błędzie, 100  
 informacje o pliku, 269  
 inicjalizacja, 40  
   pól, 198  
   tablic, 101  
   właściwości, 196  
   zmiennej, 40  
 inicjalizator, 152  
 inkrementacja, 51  
 instalacja  
   .NET Framework, 13  
   Mono, 15  
   MonoDevelop, 15  
   Visual Studio, 14  
 instrukcja  
   break, 91  
   Console.Write, 48  
   continue, 95  
   goto, 79  
   if, 68  
   if...else, 67, 69  
   if...else if, 72  
   return, 188  
   switch, 76–79

instrukcje  
   sterujące, 67  
   warunkowe, 67  
 interfejs, 199, 319, 322, 324  
   graficzny, 359  
   IDrawable, 321  
 interpolacja łańcuchów znakowych, 46

**J**

jednostki kompilacji, 126  
 język C#, 9

**K**

katalog, 259  
 klasa, 118  
   BinaryReader, 285  
   BinaryWriter, 283  
   Button, 399  
   ComboBox, 405  
   Console, 248  
   Convert, 238  
   DirectoryInfo, 259  
   FileInfo, 266  
   FileStream, 272  
   FileSystemInfo, 258  
   Form, 361  
   Kontener, 377, 384  
   Label, 397  
   MainForm, 365  
   StreamReader, 279  
   StreamWriter, 281  
   TablicaInt, 349, 350  
   TextBox, 401  
   Triangle, 307  
 klasy  
   abstrakcyjne, 309, 319  
   chronione, 165, 342  
   kontenerowe, 376  
   pochodne, 299  
   potomne, 156  
   prywatne, 165, 342  
   publiczne, 165, 342  
   statyczne, 129  
   wewnętrzne, 165, 342  
   wewnętrzne chronione, 165, 342  
   zagnieżdżone, 334  
   zewnętrzne, 344  
 klawiatura, 255  
 klawisze specjalne, 253  
 kod  
   pośredni, CIL, 12  
   źródłowy, 11

kolejność wykonywania konstruktorów, 315  
 kolory, 253  
 komentarz, 27  
   blokowy, 27  
   liniowy, 28  
   XML, 29  
 kompilacja, 11  
   just-in-time, 12  
 kompilator, 11  
   csc.exe, 12, 19  
   mcs, 23  
 komponenty graficzne, 392  
 komunikat, 392  
   o błędzie, 214  
 konflikty nazw, 328  
 konsola, 17  
 konsolidacja, 12  
 konstruktor, 145, 147  
   bezargumentowy, 199  
   domyślny, 314  
   inicjalizujący właściwość, 195  
   przyjmujący argumenty, 148  
   struktury, 199  
 konstruktory klasy  
   bazowej i potomnej, 160  
   BinaryReader, 285  
   BinaryWriter, 283  
 kontener, 346  
 kontrola typów, 346, 352  
 konwersja, 239, 293  
   typów prostych, 290, 293  
 konwersje wartości, 54

**L**

linia tekstu, 255  
 linkowanie, 12  
 lista plików, 261  
 listy  
   inicjalizacyjne, 152  
   listy rozwijane, 404, 405  
 literal null, 38  
 literały, 36  
   całkowitoliczbowe, 36  
   logiczne, 38  
   łańcuchowe, 38  
   zmiennoprzecinkowe, 37  
   znakowe, 38  
 logiczna negacja, 62

**Ł**

łańcuchy znakowe, 35, 233  
 łączenie napisów, 45

**M**

manifest, 127  
 menu, 366, 368, 395  
   Kompilacja, 21  
   rozwijane, 368  
   Tools, 21  
   z dwoma podpozycjami, 396  
 metadane, 127  
 metoda, 122  
   Draw, 311  
   DrawShape, 306, 312  
   Main, 125, 127, 136  
   ToString, 296, 297  
 metody  
   abstrakcyjne, 309  
   klasy BinaryReader, 285, 286  
   klasy BinaryWriter, 283  
   klasy Console, 249  
   klasy Convert, 238  
   klasy DirectoryInfo, 260  
   klasy FileInfo, 267  
   klasy FileStream, 272  
   klasy FileSystemInfo, 259  
   klasy Form, 362  
   klasy StreamReader, 279  
   klasy StreamWriter, 281  
   klasy string, 243  
   operujące na polu, 160  
   prywatne, 307  
   statyczne, 183  
   ustawiające pola, 132  
   wirtualne, 303, 305  
   zwracające wyniki, 125  
   zwrotne, 377  
 modyfikator sealed, 174  
 modyfikatory dostępu, 164  
 Mono, 15, 22  
 MonoDevelop, 15, 23

**N**

nawiasy klamrowe, 68  
 nazwy zmiennych, 42  
 negacja bitowa, 59  
 nieprawidłowe dziedziczenie interfejsów, 333  
 niszczenie obiektu, 154

**O**

obiekt, 118  
   generujący zdarzenie, 386  
   jako argument, 134

- obiekt
    - klasy zagnieżdżonej, 341
    - wyjątku, 191
  - obiekty klas zagnieżdżonych, 339
  - obsługa
    - błędów, 191, 203
    - kilku zdarzeń, 390
    - zdarzeń, 383, 393
  - odczyt
    - danych binarnych, 285
    - danych binarnych z pliku, 286
    - danych tekstowych, 278
    - danych z pliku, 276
    - danych z pliku tekstowego, 279
    - plików, 271
    - znaków, 248
  - odśmiecacz, 154
  - odwołanie do składowej, 292
  - okno, 359, 363, 364
    - dialogowe, 393
    - konsoli, 17
  - opcja
    - Debug, 21
    - Release, 21
  - opcje kompilatora csc, 19
  - operacje
    - arytmetyczne, 50
    - bitowe, 57
    - logiczne, 61
    - na katalogach, 259
    - na plikach, 266
    - na tablicach, 98
    - odczytu i zapisu, 274
    - przypisania, 63
    - strumieniowe, 278
  - operator
    - ., 121
    - dekrementacji, 53
    - inkrementacji, 52
    - new, 101, 106, 120
    - warunkowy, 76, 81
  - operatory
    - arytmetyczne, 50
    - bitowe, 57
    - logiczne, 61
    - porównywania, 64
    - przypisania, 63, 64
  - ostrzeżenie kompilatora, 159
- P**
- pakiet
    - .NET Framework, 12
    - Microsoft Build Tools 2015, 18
    - Visual C#, 12
    - Visual Studio, 12
    - Xamarin Studio, 15, 25
  - parametr, 131
  - pętla
    - do...while, 88
    - for, 83
    - foreach, 89
    - while, 86
  - pierwsza aplikacja, 16
  - platforma .NET, 12
  - pliki
    - cs, 16
    - pośrednie, 11
    - wykonywalne, 11
  - pobieranie
    - linii tekstu, 255
    - zawartości katalogu, 260
  - pola
    - statyczne, 184
    - tekstowe, 401, 403
  - pole
    - chronione, 169
    - prywatne, 168
    - publiczne, 166
    - sygnalizujące stan operacji, 205
    - tylko do odczytu, 175–177
  - polecenie
    - cd, 18
    - cmd, 17
  - polimorficzne wywoływanie metod, 317
  - polimorfizm, 289, 302
  - poprawność danych, 203
  - późne wiązanie, 299, 302
  - priorytety operatorów, 65
  - problem kontroli typów, 352
  - procedura obsługi zdarzenia, 384
  - programowanie obiektowe, 117, 289
  - propagacja wyjątku, 210
  - prywatne
    - klasy zagnieżdżone, 342
    - pola, 168
  - przechowywanie dowolnych danych, 350
  - przechwytywanie
    - wyjątku, 209
    - wielu wyjątków, 215, 217
    - wyjątku ogólnego, 215
  - przeciążanie
    - konstruktorów, 149
    - metod, 131, 138, 329
  - przekazywanie argumentów
    - przez referencję, 141
    - przez wartość, 140
  - przekroczenie zakresu, 55, 57



- przekształcanie współrzędnych, 171
- przerwanie
  - wykonywania pętli, 94
  - instrukcji switch, 79
- przesłanie
  - metod, 179, 296
  - pól, 182
- przeźródło nazw, 127
- przesunięcie bitowe
  - w lewo, 60
  - bitowe w prawo, 61
- przetwarzanie
  - ciągów, 242
  - znaków specjalnych, 48
- przyciski, 399
- przypisanie, 39
- publiczne pola, 167
- pusty ciąg znaków, 236

**R**

- rodzaje
  - klas wewnętrznych, 342
  - wyjątków, 212
- rozpoznawanie klawiszy specjalnych, 252
- rzeczywisty typ obiektu, 300
- rzutowanie typów obiektowych, 291–295

**S**

- sekcja finally, 228
- sekwencja ucieczki, 47
- składowe
  - klas zagnieżdżonych, 338
  - klasy zewnętrznej, 344
  - statyczne, 183
  - typu wyliczeniowego, 403
- słowo kluczowe
  - namespace, 127
  - sealed, 174
  - this, 151
  - void, 123
- specyfikatory formatów, 241
- sprawdzanie poprawności danych, 188
- stałe napisowe, 36
- standardowe wejście i wyjście, 247
- standardy C#, 10
- statyczne metody, 183
- statyczne pola, 184
- struktura, 196
  - ConsoleKeyInfo, 250, 252
  - struktura kodu, 26
  - struktura tablicy, 97
  - struktura właściwości, 187

- strumienie, 278
  - wejściowe, 278
  - wyjściowe, 278
- suma
  - bitowa, 59
  - logiczna (|), 62
  - logiczna (||), 62
- sygnalizacja błędów, 190
- system
  - plików, 258
  - wejścia-wyjścia, 233
- szkielet
  - aplikacji, 20, 25
  - klasy, 119

**Ś**

- ścieżka dostępu, 18
- środowisko uruchomieniowe, CLR, 12

**T**

- tablica, 97
- tablice
  - dwuwymiarowe, 104, 109
  - nieregularne, 111
  - tablic, 107
  - trójkątne, 114
- technologia Silverlight, 23
- tekst programu, 21
- testowanie
  - klasy, 158
  - konstruktora, 149
- tworzenie
  - delegacji, 371
  - interfejsów, 319
  - katalogów, 263
  - klas zagnieżdżonych, 334
  - menu, 366
  - obiektu, 120
  - obiektu klasy zagnieżdżonej, 342
  - okien aplikacji, 359
  - okna aplikacji, 360
  - pliku, 267
  - struktur, 196, 197
  - tablic, 98
  - tablicy dwuwymiarowej, 106
  - tablicy nieregularnej, 113
  - własnych wyjątków, 225
- typ
  - bool, 34
  - char, 34
  - ContentAlignment, 403
  - Object, 295
  - string, 35, 243

## typy

- arytmetyczne całkowitoliczbowe, 32
- arytmetyczne zmiennoprzecinkowe, 33
- danych, 31
- obiektów, 118
- odnośnikowe, 32
- proste, 32, 297
- strukturalne, 35
- uogólnione, 346, 353
- wartościowe, 32
- wyliczeniowe, 34, 403

## U

uniwersalność interfejsów, 327

uogólnianie typów, 355

uogólniona klasa, 354

uruchomienie programu, 16, 22

usuwanie

katalogów, 265

plików, 270

użycie

bloku try...catch, 208

break, 91

delegacji, 379

dyrektywy using, 129

etykiety, 398

instrukcji continue, 95

instrukcji goto, 79

instrukcji if, 68

instrukcji if...else if, 74

instrukcji switch, 77

klas zagnieżdżonych, 336

klasy, 121

klasy Convert, 239

klasy Kontener, 387

klasy Tablica, 351

komentarza blokowego, 27

komentarza liniowego, 29

komentarza XML, 30

listy rozwijanej, 405

metody zwrotnej, 377

obiektu klasy zagnieżdżonej, 340, 341

operatora dekrementacji, 53

operatora inkrementacji, 52

operatora new, 106

operatora warunkowego, 81

pętli do...while, 88

pętli foreach, 90

pola tekstowego, 402

prostej właściwości, 187

przeciążonych konstruktorów, 150

sekcji try...finally, 228, 230

struktury, 197

właściwości Length, 110

właściwości Message, 213

zdarzenia ApplicationExit, 394

## V

Visual Studio, 14, 19, 14, 19

Visual Studio Community, 12

## W

wczesne wiązanie, 302

wczytanie pojedynczego znaku, 250

wczytywanie tekstu, 255

wiązanie

czasu wykonania, 302

dynamiczne, 302

statyczne, 302

własne wyjątki, 220

właściwości, 187, 324

implementowane automatycznie, 195

klasy Button, 399

klasy ComboBox, 405

klasy Console, 248

klasy DirectoryInfo, 259

klasy FileInfo, 266

klasy FileStream, 272

klasy FileSystemInfo, 259

klasy Form, 361

klasy Label, 397

klasy TextBox, 401, 402

niezwiązane z polami, 194

struktury ConsoleKeyInfo, 250

tylko do odczytu, 192

tylko do zapisu, 193

właściwość

Length, 102, 109

Message, 213

typu ObjectCollection, 404

wnętrze klasy, 170

wprowadzanie liczb, 256

współrzędne

biegunowe, 171

kartezjańskie, 171

wybór typu projektu, 20, 24

wyjątek, 190, 207

DivideByZeroException, 213, 222, 318

IndexOutOfRangeException, 213

InvalidCastException, 301, 353

ValueOutOfRangeException, 191

wyjątki

hierarchia, 214

ponowne zgłoszenie, 223

propagacja, 210

- przechwycenie, 209
- przechwytywanie, 215
- warunkowe, 226
- własne, 220, 225
- zgłaszanie, 221
- wypełnianie tablicy, 102, 114
- wyprowadzanie danych na ekran, 43
- wyrażenia lambda, 143
  - definicja metody, 144
- wyrażenie
  - modyfikujące, 84
  - początkowe, 84, 85
  - warunkowe, 84
- wyświetlanie
  - katalogu bieżącego, 260
  - komunikatów, 392
  - liczb, 242
  - listy podkatalogów, 261
  - nazwy plików, 263
  - okna dialogowego, 392
  - napisu, 44
  - pojedynczych znaków, 235
  - wartości zmiennych, 43
  - zawartości tablicy, 105
  - znaków specjalnych, 46
- wywołanie
  - konstruktorów, 163, 313, 316
  - metody, 123
  - metody przez delegację, 374
  - polimorficzne, 304, 306
  - metod, 302
  - metod w konstruktorach, 316

**X**

Xamarin Studio, 23

**Z**

- zabronienie dziedziczenia, 174
- zagnieżdżanie
  - bloków try...catch, 218
  - instrukcji if...else, 69
  - komentarzy blokowych, 28
- zagnieżdżone pętle for, 93
- zakresy liczb, 33
- zamiana ciągów na wartości, 238
- zapis
  - danych binarnych, 283
  - danych binarnych do pliku, 284
  - danych do pliku, 274
  - danych tekstowych, 281
  - danych w pliku tekstowym, 281
  - plików, 271
  - wartości, 36
- zdarzenia, 371, 381, 389
- zdarzenie ApplicationExit, 394
- zestaw, 127
  - bibliotek, FCL, 12
- zgłaszanie
  - ponowne wyjątku, 223
  - przechwyconego wyjątku, 223
  - własnych wyjątków, 226
  - wyjątków, 221
- zintegrowane środowisko programistyczne, IDE, 15
- zmiana kolorów na konsoli, 254
- zmienna systemowa path, 18
- zmiennie, 39
  - odnośnikowe, 121
  - typów odnośnikowych, 42
- znaczniki komentarza XML, 29
- znaki, 233
  - specjalne, 35, 46, 237



# 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>



# PRAKTYCZNY KURS

## C# należy do rodziny obiektowych języków programowania.

Co to oznacza? Otóż znajduje się on w doborowym towarzystwie Javy, C++ i wielu innych języków, za pomocą których można tworzyć wspaniałe aplikacje, ale jednocześnie jest od nich odrobinę łatwiejszy. Jeśli dopiero zaczynasz przygodę z programowaniem, szybciej uda Ci się go opanować, a jeżeli jest to dla Ciebie któryś kolejny język, zrozumienie jego składni i reguł nie powinno Ci sprawić większych trudności — szczególnie jeśli do nauki wykorzystasz tę książkę.

**Ten znakomity, praktyczny podręcznik pozwoli Ci** przećwiczyć używanie i sposób działania wszystkich elementów C# — różnych typów danych, zmiennych i operatorów, instrukcji i tablic. Zobaczysz, jak korzystać z pętli i jak zachowują się obiekty. Poznasz najróżniejsze rodzaje klas, opanujesz wygodne sposoby stosowania dziedziczenia i nauczysz się obsługiwać błędy. W dalszej części książki znajdziesz zaawansowane zagadnienia programowania obiektowego i odkryjesz, jak projektować aplikacje z interfejsem graficznym. Krótko mówiąc, po starannym wykonaniu ćwiczeń będziesz w stanie zaprojektować i zbudować własną aplikację z użyciem języka C#!

- Typy danych, zmienne i instrukcje sterujące
- Tablice i pętle
- Klasy i obiekty
- Dziedziczenie
- Obsługa błędów i wyjątków
- System wejścia–wyjścia
- Polimorfizm i interfejsy
- Klasy zagnieżdżone i typy uogólnione
- Aplikacje z interfejsem graficznym

## Stwórz własną aplikację w języku C#!

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

# Helion

41171

numer katalogowy

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:

① <http://helion.pl/promocje>

Książki najchętniej czytane:

② <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

③ <http://helion.pl/nowosci>

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

ISBN 978-83-283-1456-6



9 788328 314566

Informatyka w najlepszym wydaniu

cena: 59,00 zł