

Rozpocznij przygodę z C# i platformą .NET!

Wydanie drugie
Obejmuje C# i .NET 4.0
oraz Visual Studio 2010

Rusz głową!

C#



Zarządzaj danymi,
używając LINQ

Napisz w pełni
funkcjonalną grę
zręcznościową
w stylu retro



Dowiedz się, jak metody
rozszerzające pomogły
Zuzannie nagiąć reguły
panujące w Obiekcie

Przewodnik do
nauki praktycznego
programowania
w C# i .NET



Poznaj sekrety abstrakcji
i dziedziczenia



Przekonaj się, jak Jakub
użył kolekcji generycznych,
by zapanować nad swoimi
danymi

O'REILLY®

Andrew Stellman, Jennifer Greene PSE

Helion

» Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

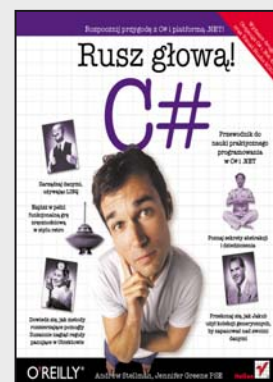
- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2011

C#. Rusz głową!

Autorzy: Andrew Stellman, Jennifer Greene PSE
Tłumaczenie: Piotr Rajca na podstawie „Head First C#.
Edycja polska” w tłumaczeniu Pawła Dyla
ISBN: 978-83-246-2953-4
Tytuł oryginału: [Head First C#: A Learner's Guide to Real-World Programming with Visual C# and .NET](#)
Format: 200×230, stron: 800



Rozpocznij przygodę z C# i platformą .NET!

- Jak tworzyć kod dla różnych platform?
- Jak przygotować środowisko pracy?
- Jak operować na dużych zbiorach danych z użyciem LINQ?

C# to jeden z języków, dzięki którym możesz pisać przenośny kod. Nie musisz się martwić o to, jakiego systemu używa Twój klient. Najważniejsze, żeby posiadał środowisko uruchomieniowe: .NET Framework, Mono lub DotGNU. Czyż nie zawsze marzyłeś o tym żeby napisać kod raz, a potem bez żadnych dodatkowych nakładów uruchamiać go na różnych platformach? Twoje marzenia właśnie się spełniają!

Dzięki tej książce, należącej do cenionej serii „Rusz głową”, opanujesz język C# w mgnieniu oka! Tylko kilkaset stron dzieli Cię od swobodnego poruszania się w kodzie napisanym w tym języku. Każda z tych stron charakteryzuje się odpowiednią dawką humoru, doskonałą przejrzystością oraz perfekcyjnie przekazaną wiedzą. Czego się nauczysz? Przede wszystkim dowiesz się, jak stworzyć działający program w 10 minut. Następnie poznasz elementy programowania obiektowego – takie pojęcia jak hermetyzacja czy dziedziczenie nie będą Ci obce! Kolejne strony przynoszą szeroki zakres wiedzy dotyczący operacji na plikach, obsługi wyjątków oraz tworzenia interfejsu użytkownika. Wreszcie poznasz język LINQ służący do efektywnego operowania na zbiorach danych. „C#. Rusz głową!” to idealna propozycja dla wszystkich czytelników chcących rozpocząć przygodę z językiem C# oraz platformą .NET.

- Przygotowanie środowiska pracy, zapoznanie z Visual Studio
- Wsparcie Visual Studio dla programisty
- Anatomia programu
- Praca z debuggerem
- Pętle, instrukcje warunkowe
- Elementy programowania obiektowego
- Typy zmiennych
- Referencje
- Tablice
- Hermetyzacja obiektów
- Implementacja interfejsów
- Typy wyczerpieniowe
- Operowanie strumieniami danych
- Obsługa wyjątków
- Wykorzystanie języka LINQ do operacji na bazach danych i dużych zbiorach informacji
- Tworzenie interfejsu użytkownika

Zobacz, jakie możliwości kryje język C#. To nie jest trudne!

Spis treści (skrótowy)

Wstęp	29
1 Zwiększ wydajność przy pomocy C#: <i>Aplikacje Visual Studio w 10 minut lub mniej</i>	41
2 To tylko kod: <i>Pod maską</i>	79
3 Obiekty: zorientuj się! <i>Tworzenie kodu ma sens</i>	121
4 Typy i referencje: <i>Jest 10:00. Czy wiesz, gdzie są Twoje dane?</i>	159
Laboratorium C# numer 1: <i>Dzień na wyścigach</i>	201
5 Hermetyzacja: <i>Co ma być ukryte... niech będzie ukryte</i>	211
6 Dziedziczenie: <i>Drzewo genealogiczne Twoich obiektów</i>	247
7 Interfejsy i klasy abstrakcyjne: <i>Klasy, które dotrzymują swoich obietnic</i>	299
8 Typy wyliczeniowe i kolekcje: <i>Przechowywanie dużej ilości danych</i>	355
Laboratorium C# numer 2: <i>Wyprawa</i>	411
9 Odczyt i zapis plików: <i>Zapisz tablice bajtów, zapisz świat</i>	433
10 Obsługa wyjątków: <i>Gaszenie pożarów nie jest już popularne</i>	487
11 Zdarzenia i delegaty: <i>Co robi Twój kod, kiedy nie patrzysz</i>	529
12 Powtórka i pokaz: <i>Wiedza, moc i tworzenie ciekawych rzeczy</i>	563
13 Kontrolki i grafika: <i>Upiększ to</i>	611
14 Kapitan Wspaniały: <i>Śmierć obiektu</i>	669
15 LINQ: <i>Przejmij kontrolę nad danymi</i>	707
Laboratorium C# numer 3: <i>Invaders</i>	735
Dodatek A <i>Pozostałości: 11 najważniejszych rzeczy, które chcieliśmy umieścić w tej książce</i>	757
Skorowidz	791

Spis treści (z prawdziwego zdarzenia)



Wstęp

Przygotuj się na C#. Właśnie sobie siedzisz i próbujesz się czegoś nauczyć, ale mózg wciąż powtarza Ci, że cała ta nauka *nie jest ważna*. Twój umysł mówi: „Lepiej wyjdź z pokoju i zajmij się ważniejszymi sprawami, takimi jak to, których dzikich zwierząt unikać, oraz to, że strzelanie z łuku na golasa nie jest dobrym pomysłem”. W jaki sposób oszukać mózg, tak aby myślał, że Twoje życie naprawdę zależy od nauki C#?

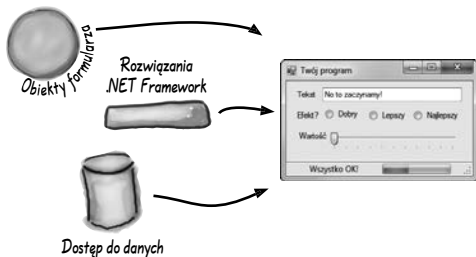
Dla kogo jest ta książka?	30
Wiemy, o czym myślisz	31
Metapoznanie: myślenie o myśleniu	33
Zmuś swój mózg do posłuszeństwa	35
Przeczytaj to	37
Grupa korektorów technicznych	38
Podziękowania	39

Zwiększ wydajność przy pomocy C#

1

Aplikacje Visual Studio w 10 minut lub mniej

Czy chcesz tworzyć wspaniałe programy naprawdę szybko? Wraz z C# dostajesz do ręki **potężny język programowania** i wartościowe narzędzie. Dzięki **Visual Studio IDE** do historii przejdą sytuacje, w których musiałeś pisać jakiś nędzny kod, by ponownie zapewnić prawidłowe działanie przycisku. I to nie wszystko. Dodatkowo będziesz mógł **skupić się na faktycznym wykonywaniu swojej pracy**, zamiast starać się zapamiętać, który parametr metody odpowiadał za *nazwę przycisku*, a który za *wyświetlany na nim tekst*. Brzmi zachęcająco? Przewróć zatem stronę i przystąpmy do programowania.



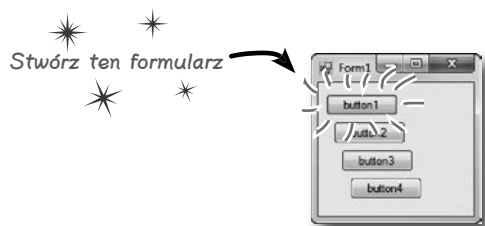
Dlaczego powinieneś uczyć się C#	42
C# oraz Visual Studio ułatwiają wiele czynności	43
Pomóż dyrektorowi naczelnemu zrezygnować z papieru	44
Sprawdź potrzeby Twoich użytkowników, zanim zaczniesz tworzyć program	45
Oto program, który zamierzasz stworzyć	46
Co robisz w Visual Studio	48
Co Visual Studio robi za Ciebie	48
Stwórz interfejs użytkownika	52
Za kulisami Visual Studio	54
Dodaj coś do automatycznie wygenerowanego kodu	55
Potrzebujemy bazy danych do przechowywania naszych informacji	58
IDE utworzyło bazę danych	59
SQL jest swoim własnym językiem	59
Tworzenie tabeli dla listy kontaktowej	60
Zakończ tworzenie tabeli	65
Wstaw dane z kart do bazy	66
Połącz formularz z bazą danych, korzystając ze źródeł danych	68
Dodaj kontrolki powiązane z bazą danych do formularza	70
Jak zamienić TWOJĄ aplikację w aplikację WSZYSTKICH	75
Przełącz aplikację innym użytkownikom	76
Jeszcze nie skończyłeś: przetestuj instalację	77
Stworzyłeś pełnowartościową aplikację bazodanową	78

2

To tylko kod

Pod maską

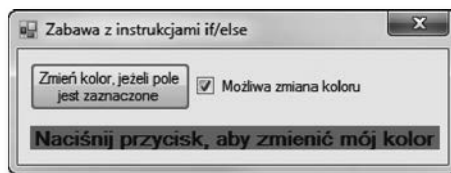
Jesteś programistą, nie jedynie użytkownikiem IDE. IDE może wykonać za Ciebie wiele pracy, ale na razie jest to wszystko, co może dla Ciebie zrobić. Oczywiście, istnieje wiele **powtarzalnych czynności** podczas pisania aplikacji i IDE okazuje się tu bardzo pomocne. Praca z nim to jednak *dopiero początek*. Możesz wycisnąć ze swoich programów znacznie więcej — **pisanie kodu C#** to właśnie droga, która doprowadzi Cię do tego celu. Jak tylko osiągniesz mistrzowski poziom w kodowaniu, nie będzie *żadnej* rzeczy, której Twój program nie umiałby zrobić.



Za każdym razem, kiedy stworzysz nowy program, definiujesz dla niego przestrzeń nazw. W ten sposób jego kod jest odseparowany od innych klas platformy .NET.

Klasy zawierają **fragmenty** kodu Twojego programu (choć istnieją także bardzo małe aplikacje składające się z tylko jednej klasy).

Klasa posiada jedną lub więcej metod. Twoje metody zawsze będą umieszczane **wewnątrz klas**, a każda z nich będzie się składała z instrukcji i wyrażeń — jak te, które do tej pory widziałeś.



Kiedy robisz to...	80
...IDE robi to	81
Skąd się biorą programy	82
IDE pomaga Ci kodować	84
Kiedy zmieniasz coś w IDE, zmieniasz także swój kod	86
Anatomia programu	88
Twój program wie, gdzie zacząć	90
W tej samej przestrzeni nazw mogą być dwie klasy	97
Twoje programy używają zmiennych do pracy z danymi	98
C# używa znanych symboli matematycznych	100
Użyj debugera by zobaczyć jak zmieniają się wartości zmiennych	101
Pętle wykonują czynność wielokrotnie	103
Kodowanie czas zacząć	104
Instrukcje if/else podejmują decyzje	105
Ustal warunki i sprawdź, czy są prawdziwe	106

Obiekty: zorientuj się!

3

Tworzenie kodu ma sens

Każdy pisany przez Ciebie program rozwiązuje jakiś problem.

Rozpoczynając pisanie programu zawsze warto zacząć od zastanowienia się, jaki *problem* ma on rozwiązywać. Pozwalają one tworzyć strukturę kodu tak, by odpowiadała ona rozwiązywanemu problemowi, dzięki czemu będziesz mógł *skoncentrować się na samym problemie*, a nie na mechanice tworzenia kodu. Prawidłowe użycie obiektów spowoduje, że proces pisania kodu stanie się bardziej *intuicyjny*, a jego późniejsza analiza i modyfikacja — znacznie łatwiejsze.

W jaki sposób Maciek myśli o swoich problemach 122

W jaki sposób system nawigacyjny w samochodzie Maćka rozwiązuje jego problemy 123

Klasa Navigator napisana przez Maćka posiada metody do ustalania i modyfikacji tras 124

Wykorzystaj to, czego się nauczyłeś, do napisania prostego programu używającego klas 125

Maciek może użyć obiektów do rozwiązania swojego problemu 128

Używasz klasy do utworzenia obiektu 129

Kiedy tworzysz obiekt na podstawie klasy, to taki obiekt nazywamy instancją klasy 130

Lepsze rozwiązanie... uzyskane dzięki obiektom! 131

Instancja używa pól do przechowywania informacji 136

Stwórzmy kilka instancji! 137

Dzięki za pamięć 138

Co Twój program ma na myśli 139

Możesz używać nazw klas i metod w celu uczynienia kodu bardziej intuicyjnym 140

Nadaj swojej klasie naturalną strukturę 142

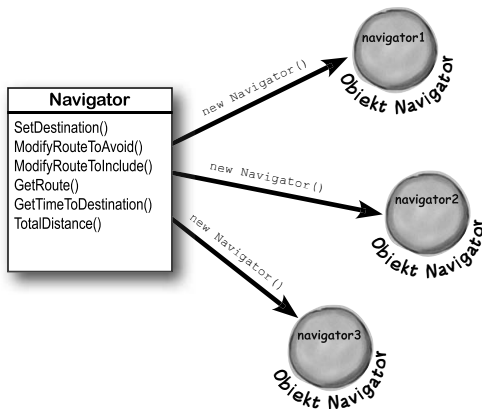
Diagramy klas pozwalają w sensowny sposób zorganizować klasy 144

Utwórz klasę do pracy z kilkoma facetami 148

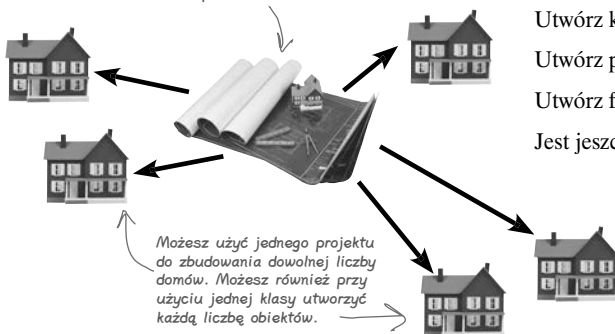
Utwórz projekt dla facetów 149

Utwórz formularz do interakcji z facetami 150

Jest jeszcze prostszy sposób inicjalizacji obiektów 153



Kiedy definiujesz klasę, definiujesz także jej metody, podobnie jak projekt definiuje układ pomieszczeń w domu.



Możesz użyć jednego projektu do zbudowania dowolnej liczby domów. Możesz również przy użyciu jednej klasy utworzyć każdą liczbę obiektów.

Typy i referencje

4

Jest 10:00. Czy wiesz, gdzie są Twoje dane?

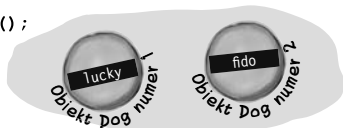
Typ danych, baza danych, dane komandora porucznika... wszystko to są ważne rzeczy. Bez danych Twoje programy są beużyteczne. Potrzebujesz informacji dostarczanych przez użytkowników. Na jej podstawie wyszukujesz lub tworzysz nową **informację** i zwracasz ją użytkownikom. W rzeczywistości prawie wszystko, co robisz podczas programowania, sprowadza się do **pracy z danymi** w taki czy w inny sposób. W tym rozdziale dowiesz się o różnych aspektach **typów danych** C#, nauczysz się pracować z danymi w programie, a nawet odkryjesz kilka pilnie strzeżonych sekretów o **obiektach** (*pssst... obiekty to także dane*).

Typ zmiennej określa rodzaj danych, jakie zmienna może przechowywać	160
Zmienna jest jak kubek z danymi	162
10 kilogramów danych w pięciokilogramowej torebce	163
Nawet wtedy, gdy liczba ma prawidłowy rozmiar, nie możesz przypisać jej do każdej zmiennej	164
Kiedy rzutujesz wartość, która jest zbyt duża, C# dopasowuje ją automatycznie	165
C# przeprowadza niektóre rzutowania automatycznie	166
Kiedy wywołujesz metodę, zmienne muszą pasować do typów parametrów	167
Połączenie = z operatorem	172
Także obiekty używają zmiennych	173
Korzystaj ze swoich obiektów za pomocą zmiennych referencyjnych	174
Referencje są jak etykiety do Twoich obiektów	175
Jeżeli nie ma już żadnej referencji, Twoje obiekty są usuwane z pamięci	176
Referencje wielokrotne i ich efekty uboczne	177
Dwie referencje oznaczają DWA sposoby na zmianę danych obiektu	182
Specjalny przypadek: tablice	183
Witamy w barze Niechlujny Janek — najtańsze kanapki w mieście!	185
Obiekty używają referencji do komunikacji między sobą	187
Tam, gdzie obiektów jeszcze nie było	188
Napisz grę w literki	193

```
Dog fido;
Dog lucky = new Dog();
```



```
fido = new Dog();
```



```
lucky = null;
```



Laboratorium C# numer 1

Dzień na wyścigach

Janek, Bartek i Arek uwielbiają chodzić na tor wyścigowy, ale ciągła utrata pieniędzy powoduje u nich frustrację. Potrzebują symulatora, aby mogli określić zwycięzcę, zanim wyłożą pieniądze na zakłady. Jeśli dobrze wywiążesz się z zadania, będziesz miał procenty z ich wygranych.

Specyfikacja: stwórz symulator wyścigów	202
Końcowy produkt	210



Hermetyzacja

5

Co ma być ukryte... niech będzie ukryte

Czy kiedykolwiek marzyłeś o odrobinie prywatności? Czasami Twoje obiekty czują się tak samo. Na pewno nie lubisz sytuacji, w których ktoś, komu nie ufasz, czyta Twój pamiętnik lub przegląda wykazy Twoich operacji bankowych. Dobre obiekty nie pozwalają *innym* obiektom na oglądanie swoich pól. W tym rozdziale nauczysz się wykorzystywać potęgę hermetyzacji. Sprawisz, że dane obiektów będą prywatne i dodasz metody, które pozwolą Ci na zabezpieczenie dostępu do danych.



ciaAgent

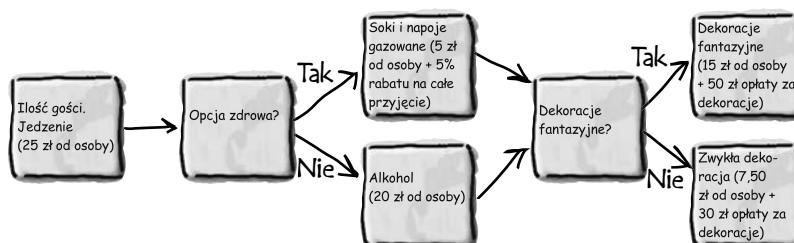


kgbAgent



mi5Agent

Krystyna planuje przyjęcia	212
Co powinien robić program szacujący?	213
Jazda próbna Krystyny	218
Każda opcja powinna być obliczana osobno	220
Bardzo łatwo przez przypadek źle skorzystać z obiektów	222
Hermetyzacja oznacza, że niektóre dane w klasie są prywatne	223
Użyj hermetyzacji w celu kontroli dostępu do metod i pól Twojej klasy	224
Ale czy jego prawdziwa tożsamość jest NAPRAWDĘ chroniona?	225
Dostęp do prywatnych pól i metod można uzyskać tylko z wnętrza klasy	226
Hermetyzacja utrzymuje Twoje dane w nieskazitelny stan	234
Właściwości sprawiają, że hermetyzacja będzie łatwiejsza	235
Stwórz aplikację do przetestowania klasy Farmer	236
Użyj automatycznych właściwości do ukończenia klasy	237
Co wtedy, gdy chcemy zmienić pole mnożnika żywienia?	238
Użyj konstruktora do inicjalizacji pól prywatnych	239



Dziedziczenie

6

Drzewo genealogiczne Twoich obiektów

Czasami CHCIAŁBYŚ być dokładnie taki sam jak Twoi rodzice. Czy kiedykolwiek natknąłeś się na obiekt, który robiłby *prawie* wszystko, czego byś sobie od niego życzył? Czy kiedykolwiek znalazłeś się w takiej sytuacji, że gdybyś *zmienił dosłownie kilka rzeczy*, obiekt byłby doskonały? Cóż, to tylko jeden z wielu powodów, które sprawiają, że **dziedziczenie** zalicza się do najważniejszej koncepcji i technik w języku C#. Kiedy skończysz czytać ten rozdział, dowiesz się jak **rozszerzać** obiekty, by móc wykorzystać ich zachowania i jednocześnie dysponować **elastycznością**, która pozwoli Ci te zachowania modyfikować. Unikniesz **wielokrotnego pisania kodu**, **przedstawisz prawdziwy świat** znacznie dokładniej, a w efekcie otrzymasz kod **łatwiejszy do zarządzania**.

Krystyna organizuje także przyjęcia urodzinowe	248
Potrzebujemy klasy BirthdayParty	249
Stwórz program Planista przyjęć w wersji 2.0	250
Kiedy klasy używają dziedziczenia, kod musi być napisany tylko raz	258
Zbuduj model klasy, rozpoczynając od rzeczy ogólnych i przechodząc do bardziej konkretnych	259
W jaki sposób zaprojektowałbyś symulator zoo?	260
Użyj dziedziczenia w celu uniknięcia zwielenokrotniania kodu w klasach potomnych	261
Pomyśl, w jaki sposób pogrupować zwierzęta	263
Stwórz hierarchię klas	264
Każda klasa pochodna rozszerza klasę bazową	265
Klasa pochodna może przesłaniać odziedziczone metody w celu ich modyfikacji lub zmiany	270
W każdym miejscu, gdzie możesz skorzystać z klasy bazowej, możesz zamiast niej użyć jednej z jej klas pochodnych	271
Klasa pochodna może ukrywać metody klasy bazowej	278
Używaj override i virtual by dziedziczyć zachowania	280
Teraz jesteś już gotowy do dokończenia zadania Krystyny	284
Stwórz system zarządzania ulem	289
Najpierw stworzysz system podstawowy	290
Użyj dziedziczenia, aby rozszerzyć system zarządzania pszczołami	294



Interfejsy i klasy abstrakcyjne

7

Klasy, które dotrzymują swoich obietnic

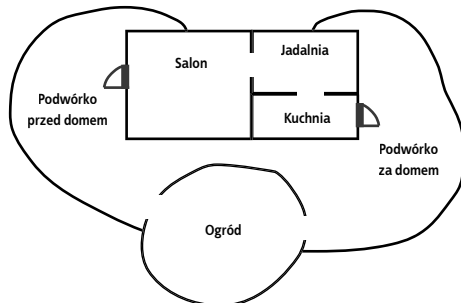
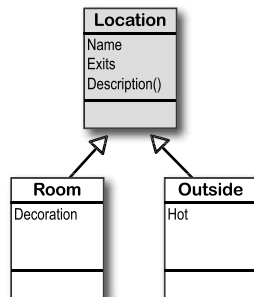
Czyny potrafią powiedzieć więcej niż słowa. Czasami potrzebujesz pogrupować swoje obiekty na podstawie tego, **co robią**, zamiast tego, po jakiej klasie dziedziczą. To jest moment, w którym należy powiedzieć o **interfejsach**. Pozwalają one na pracę z każdą klasą, która jest w stanie wykonać daną czynność. Z **wielkimi możliwościami przychodzą wielkie obowiązki** i każda klasa, która implementuje interfejs, musi **wypełnić wszystkie swoje...** albo kompilator połamie Ci kolana, zrozumiałeś?

* Dziedziczenie

* Abstrakcja

* Hermetyzacja

* Polimorfizm



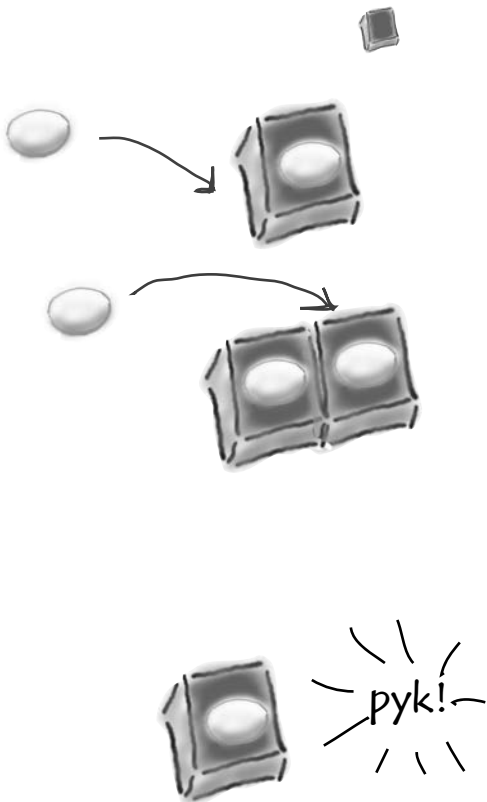
Wróćmy do pszczelej korporacji	300
Możemy użyć dziedziczenia do utworzenia klas dla różnych typów pszczół	301
Interfejs daje klasie do zrozumienia, że musi ona zaimplementować określone metody i właściwości	302
Użyj słowa kluczowego interface do zdefiniowania interfejsu	303
Klasy implementujące interfejsy muszą zawierać WSZYSTKIE ich metody	305
Nie możesz utworzyć instancji interfejsu, ale możesz uzyskać jego referencję	308
Referencje interfejsów działają tak samo jak referencje obiektów	309
Za pomocą „is” możesz sprawdzić, czy klasa implementuje określony interfejs	310
Interfejsy mogą dziedziczyć po innych interfejsach	311
Rzutowanie w górę działa w odniesieniu do obiektów i interfejsów	315
Rzutowanie w dół pozwala zamienić urządzenie z powrotem w ekspres do kawy	316
Rzutowanie w górę i w dół działa także w odniesieniu do interfejsów	317
Jest coś więcej niż tylko public i private	321
Modyfikatory dostępu zmieniają widoczność	322
Obiekty niektórych klas nigdy nie powinny być tworzone	325
Klasa abstrakcyjna jest jak skrzyżowanie klasy i interfejsu	326
Metoda abstrakcyjna nie ma ciała	329
Polimorfizm oznacza, że jeden obiekt może przyjmować wiele różnych postaci	337

Typy wyliczeniowe i kolekcje

8

Przechowywanie dużej ilości danych

Z deszczu pod rynną. W rzeczywistym świecie nie musisz się zwykle zajmować danymi w małych ilościach i w niewielkich fragmentach. Nie, Twoje dane przychodzą do Ciebie w **grupach, stosach, pękach, kopach**. Potrzebujesz jakiegoś potężnego narzędzia do ich zorganizowania. Nadszedł czas, aby przedstawić **kolekcje**. Pozwalają one **przechowywać i sortować dane, a także zarządzać** wszystkimi danymi, które Twój program musi przeanalizować. W ten sposób możesz myśleć o pisaniu programów do pracy z danymi, a samo ich przechowywanie zostawić kolekcjom.



Łańcuchy znaków nie zawsze sprawdzają się przy kategoryzowaniu danych	356
Typy wyliczeniowe pozwalają Ci wyliczyć prawidłowe wartości	357
Typy wyliczeniowe pozwalają na reprezentowanie liczb za pomocą nazw	358
Możesz użyć tablicy, aby stworzyć talię kart...	361
Listy są bardziej elastyczne niż tablice	364
Typy generyczne mogą przechowywać każdy typ	368
Inicjalizatory kolekcji działają tak samo jak inicjalizatory obiektu	372
Stwórzmy listę kaczek	373
Listy są proste, ale SORTOWANIE może być skomplikowane	374
IComparable<T> pomoże Ci posortować listę kaczek	375
Użyj interfejsu IComparer, aby powiedzieć liście, jak ma sortować	376
Utwórz instancję obiektu porównującego	377
IComparer może wykonywać złożone porównania	378
Przesłonięcie metody ToString() pozwala obiektom przedstawiać się	381
Zmień pętlę foreach tak, by obiekty Duck i Card same się opisywały	382
Używając IEnumerable możesz rzutować całą listę w górę	384
Możesz tworzyć własne przeciążone metody	385
Wybrane funkcjonalności słownika	392
Napisz program korzystający ze słownika	393
I jeszcze WIĘCEJ typów kolekcji...	405
Kolejka działa według reguły: pierwszy przyszedł, pierwszy wyszedł	406
Stos działa według reguły: ostatni przyszedł, pierwszy wyszedł	407

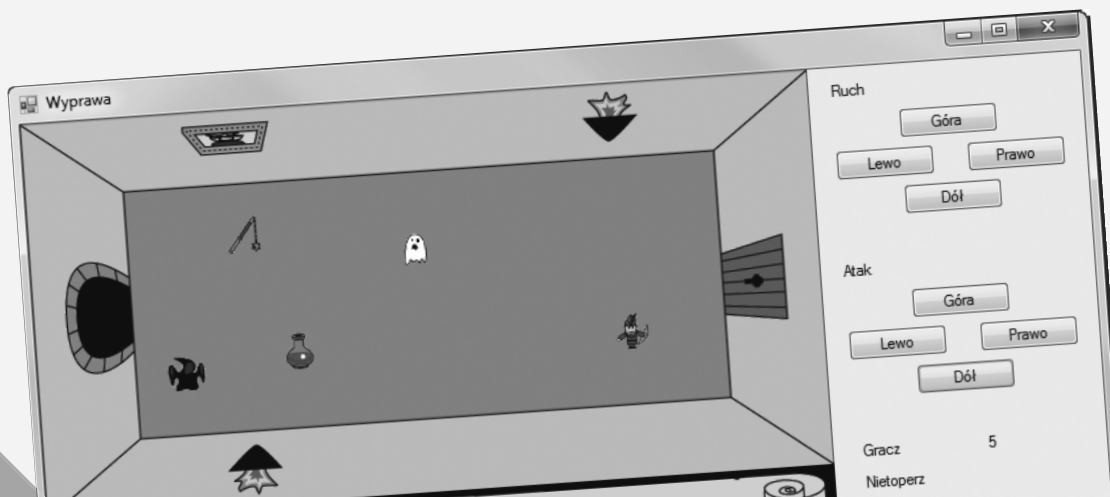
Laboratorium C# numer 2

Wyprawa

Twoim zadaniem jest stworzenie gry przygodowej, w której potężny wojownik wyrusza na misję i dzielnie walczy, poziom za poziomem, ze śmiertelnie niebezpiecznymi wrogami. Stworzysz system turowy. Oznacza to, że najpierw gracz wykonuje jeden ruch, a następnie ruch wykonuje przeciwnik. Gracz może przesunąć się lub zaatakować; potem możliwość ruchu i ataku dostaje każdy z wrogów. Gra toczy się do czasu, aż gracz pokona wszystkich przeciwników na wszystkich siedmiu poziomach lub zginie.

Specyfikacja: stwórz grę przygodową 412

Zabawa dopiero się zaczyna! 432



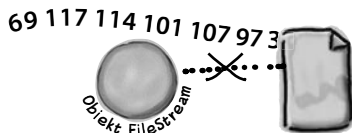
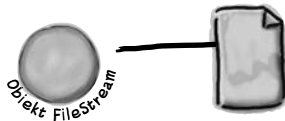
Odczyt i zapis plików

Zapisz tablice bajtów, zapisz świat

9

Czasami opłaca się być trwałym. Do tej pory wszystkie programy były krótkotrwałe. Uruchamiały się, działały przez chwilę i były zamykane. Czasami nie jest to wystarczające, zwłaszcza jeżeli zajmujesz się ważnymi danymi. Musisz mieć możliwość **zapisania swojej pracy**. W tym rozdziale pokażemy sposób **zapisywania danych do pliku**, a następnie **wczytania tych informacji z powrotem** do programu. Dowiesz się co nieco o **klasach strumieni** .NET i zetkniesz się z tajemnicami systemów **szesnastkowego** i **dwójkowego**.

C# używa strumieni do zapisu i odczytu danych	434
Różne strumienie zapisują i odczytują różne rzeczy	435
FileStream zapisuje bajty do pliku	436
W jaki sposób zapisać tekst do pliku w trzech prostych krokach	437
Zapis i odczyt wymaga dwóch obiektów	441
Dane mogą przechodzić przez więcej niż jeden strumień	442
Użyj wbudowanych obiektów do wyświetlenia standardowych okien dialogowych	445
Okna dialogowe są kolejnymi kontrolkami .NET	446
Okna dialogowe także są obiektami	447
Używaj wbudowanych klas File oraz Directory do pracy z plikami i katalogami	448
Używaj okien dialogowych do otwierania i zapisywania plików	451
Dzięki IDisposable obiekty usuwane są prawidłowo	453
Unikaj błędów systemowych, korzystając z instrukcji using	454
Zapisywanie danych do plików wymaga wielu decyzji	460
Użyj instrukcji switch do wyboru właściwej opcji	461
Serializacja pozwala Ci zapisywać lub odczytywać całe obiekty na raz	468
.NET automatycznie konwertuje tekst do postaci Unicode	473
C# może użyć tablicy bajtów do przesyłania danych	474
Pliki utworzone dzięki serializacji można także zapisywać i odczytywać ręcznie	477
Praca z plikami binarnymi może być skomplikowana	479
StreamReader i StreamWriter będą do tego odpowiednie	481
Użyj Stream.Read() do odczytywania bajtów ze strumienia	482



Obsługa wyjątków

10

Gaszenie pożarów nie jest już popularne

Programiści nie mają być strażakami. Pracowałeś jak wół, przebrnąłeś przez dokumentację techniczną i kilka zajmujących książek *Rusz Głową!*, wspiąłeś się na szczyt swoich możliwości: jesteś **mistrzem programistów**. W dalszym ciągu musisz jednak odrywać się od pracy, ponieważ **program wyłącza się lub nie zachowuje się tak, jak powinien**. Nic nie wybija Cię z rytmu tak, jak obowiązek naprawienia dziwnego błędu... Z **obsługą wyjątków** możesz jednak napisać kod, który **poradzi sobie z pojawiającymi się problemami**. Jest nawet lepiej, możesz bowiem zareagować na ich pojawienie się i sprawić, że wszystko **będzie dalej działało**.

Damian potrzebuje swoich wymówek, aby być mobilnym	488
Kiedy program zgłasza wyjątek, .NET tworzy obiekt Exception	492
Wszystkie obiekty wyjątków dziedziczą po Exception	496
Debugger pozwala Ci wysledzić wyjątki w kodzie i zapobiec im	497
Użyj debugera wbudowanego w IDE, aby znaleźć problem w programie do zarządzania wymówkami	498
Obsłuż wyjątki za pomocą try i catch	503
Co się stanie, jeżeli wywoływana metoda będzie niebezpieczna?	504
Użyj debugera do prześledzenia przepływu w blokach try/catch	506
Jeśli posiadasz kod, który ZAWSZE musi zostać wykonany, zastosuj finally	508
Jedna klasa zgłasza wyjątek, inna klasa go wyłapuje	515
Pszczoly potrzebują wyjątku OutOfHoney	516
Łatwy sposób na uniknięcie licznych problemów: using umożliwia Ci stosowanie try i finally za darmo	519
Unikanie wyjątków: zaimplementuj IDisposable, aby przeprowadzić własne procedury sprzątnięcia	520
Najgorszy z możliwych bloków catch: komentarze	522
Tymczasowe rozwiązania są dobre (tymczasowo)	523
Kilka wskazówek dotyczących obsługi wyjątków	524
Damian w końcu pojechał na urlop...	527



Zdarzenia i delegaty

11

Co robi Twój kod, kiedy nie patrzysz

Twoje obiekty zaczynają myśleć o sobie. Nie możesz zawsze kontrolować tego, co robią Twoje obiekty. Czasami różne rzeczy... zdarzają się. Kiedy to następuje, chciałbyś, aby Twoje obiekty były wystarczająco sprytnie i odpowiednio **reagowały**. To miejsce, w którym do akcji wkraczają zdarzenia. Jeden obiekt *udostępnia* zdarzenie, inny je *obsługuje* i wszystko pracuje razem, aby całość działała sprawnie. Jest to wspaniałe, o ile nie chcesz, by Twój obiekt mógł kontrolować kto będzie mógł nasłuchiwać jego zdarzeń. Wtedy bardzo pomocne okazują się **funkcje zwrotne**.

Czy kiedykolwiek marzyłeś o tym, aby Twoje obiekty potrafiły samodzielnie myśleć?	530
Ale skąd obiekt WIE, że ma odpowiedzieć?	530
Kiedy wystąpi ZDARZENIE... obiekty nasłuchują	531
Jeden obiekt wywołuje zdarzenie, inne nasłuchują...	532
Potem inne obiekty obsługują zdarzenie	533
Łącząc punkty	534
IDE automatycznie tworzy za Ciebie procedury obsługi zdarzeń	538
Ogólny typ EventHandlerer pozwala definiować własne typy zdarzeń	544
Wszystkie formularze, które utworzyłeś, używają zdarzeń	545
Jedno zdarzenie, wiele procedur obsługi	546
Połączenie nadawców zdarzenia z jego odbiorcami	548
Delegat ZASTĘPUJE właściwą metodę	549
Delegat w akcji	550
Każdy obiekt może subskrybować publiczne zdarzenie...	553
Użyj funkcji zwrotnej, by wiedzieć kto nasłuchuje	554
Funkcje zwrotne są jedynie sposobem używania delegatów	556

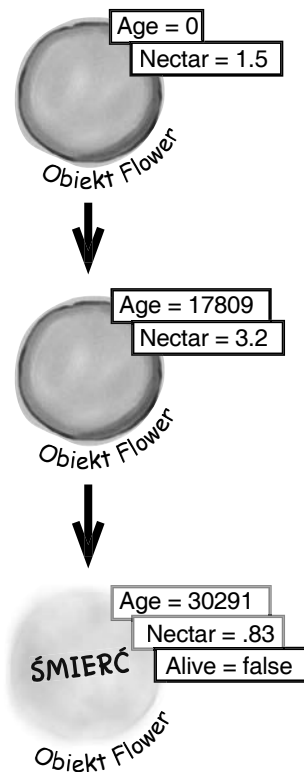


Powtórka i pokaz

12

Wiedza, moc i tworzenie ciekawych rzeczy

Uczenie się nie jest dobre, dopóki czegoś nie ZBUDUJESZ. Dopóki nie napiszesz kodu, który działa, nie będziesz miał pewności czy na pewno zrozumiałeś najtrudniejsze zagadnienia języka C#. W tym rozdziale mamy zamiar sprawdzić czego się nauczyłeś. Przedstawimy w nim także kilka nowych zagadnień, którymi już niebawem się zajmiemy. Wszystko to będzie stanowił pierwszy etap prac nad **naprawdę złożoną aplikacją**, która ma Ci pokazać, czy naprawdę dobrze opanowałeś zagadnienia opisywane w poprzednich rozdziałach. A zatem przygotuj się... nadszedł czas **pisania programu!**

Życie i śmierć kwiatów

Przebyłeś długą drogę	564
Zajmowaliśmy się także pszczołami	565
Architektura symulatora ula	566
Budowanie symulatora ula	567
Życie i śmierć kwiatów	571
Teraz potrzebujemy klasy Bee	572
PPBP (Programiści Przeciwno Bezdomnym Pszczołom)	576
Ul działa na miód	576
Wypełnianie klasy Hive	580
Metoda Go() klasy Hive	581
Jesteśmy gotowi na stworzenie świata	582
Tworzymy system turowy	583
Oto kod klasy World	584
Uczenie pszczół zachowań	590
Główny formularz wywołuje Go() dla całego świata	592
Możemy użyć obiektu World do pobrania statystyk	593
Zegary sygnalizują zdarzenia wielokrotnie	594
Za kulisami zegar używa zdarzeń	595
Pracujemy z grupami pszczół	602
Kolekcje kolekcjonują... DANE	603
LINQ ułatwia pracę z danymi w kolekcjach i bazach danych	605
Ostatnie wyzwanie: Otwórz i Zapisz	607

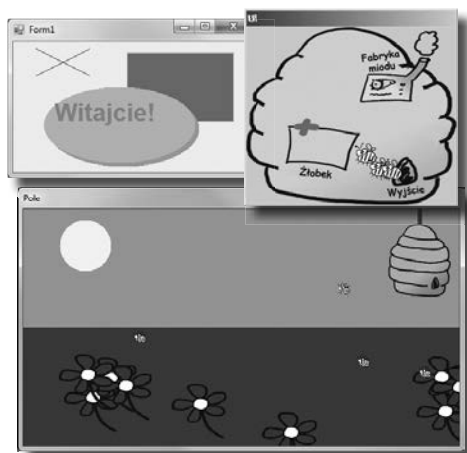
Kontrolki i grafika

13

Upiększ to

Czasami musisz wziąć sprawy grafiki we własne ręce. Przez większość czasu polegałszy na kontrolkach i w naszych aplikacjach korzystaliśmy z ich możliwości obsługi grafiki. Czasami to wszystko nie wystarcza — na przykład wtedy, gdy chcesz **animować obrazek**. Jeśli już przejdziesz do animacji, będziesz musiał dla programu .NET **stworzyć własne kontrolki**, być może dodając **podwójne buforowanie**. Czasem nawet będziesz zmuszony do **rysowania bezpośrednio na formularzach**. Wszystko to zaczyna się od obiektu **Graphics**, obiektów **Bitmap** i determinacji, aby zburzyć dotychczasowy porządek w zarządzaniu grafiką.

Cały czas do interakcji z programami używałeś kontroltek	612
Kontrolki formularza są tylko obiektami	613
Użyj kontroltek do animacji symulatora ula	614
Dodaj do projektu rendering	616
Kontrolki są dobrze dostosowane do wyświetlania różnych elementów wizualnych	618
Stwórz swoją pierwszą animowaną kontrolkę	621
Utwórz przycisk, aby dodać BeeControl do formularza	624
Twoje kontrolki także muszą usuwać swoje kontrolki!	625
UserControl to dobry sposób na tworzenie kontroltek	626
Mechanizm renderujący używa BeeControl do rysowania animowanych pszczoł na formularzu	628
Dodaj do projektu formularze reprezentujące ul i pole	630
Stwórz klasę Renderer	631
Zmieniłeś rozmiar bitmap przy pomocy obiektu Graphics	640
Zasoby Twoich obrazków przechowywane są w postaci obiektów Bitmap	641
Użyj System.Drawing, by samemu PRZEJĄĆ KONTROLĘ nad grafiką	642
30-sekundowa podróż do świata tajemnic grafiki GDI+	643
Użyj Graphics, aby na formularzu narysować obrazek	644
Klasa Graphics może usunąć problem przezroczystości...	649
Użyj zdarzenia Paint, aby grafika była mocno związana z formularzem	650
Bliższe spojrzenie na sposób rysowania formularzy i kontroltek	653
Podwójne buforowanie czyni animację bardziej płynną	656
Użyj obiektu Graphics i procedury obsługi zdarzenia do drukowania	662



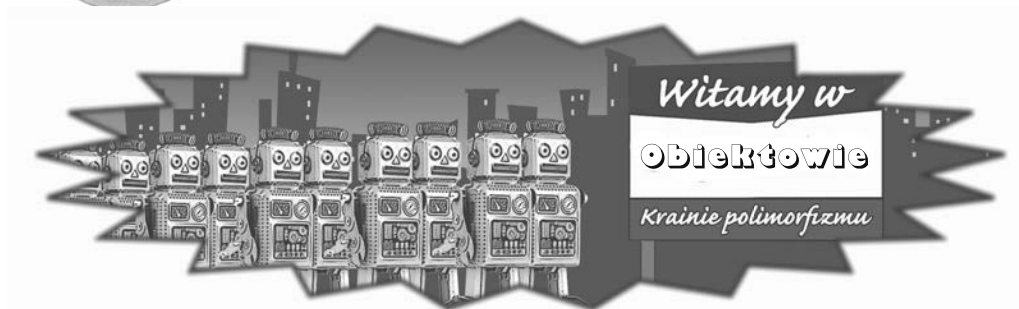
14

Kapitan Wspaniały

Śmierć obiektu



Twoją ostatnią szansą na ZROBIENIE czegoś... jest użycie finalizatora	676
Kiedy DOKŁADNIE wywołany jest finalizator?	677
Dispose() działa z using, a finalizatory działają z mechanizmem oczyszczania pamięci	678
Finalizatory nie mogą polegać na stabilności	680
Spraw, aby obiekt serializował się w Dispose()	681
Struktura jest podobna do obiektu...	685
... ale nie jest obiektem	685
Wartości są kopiowane, referencje są przypisywane	686
Stos i sterta: więcej na temat pamięci	689
Używaj parametrów wyjściowych by zwracać z metody więcej niż jedną wartość	692
Przekazuj referencje, używając modyfikatora ref	693
Używaj parametrów opcjonalnych, by określać wartości domyślne	694
Jeśli musisz używać wartości pustych, stosuj typy, które je akceptują	695
Typy akceptujące wartości puste poprawiają odporność programów	696
Kapitan Wspaniały... nie tak bardzo	699
Metody rozszerzające zwiększają funkcjonalność ISTNIEJĄCYCH klas	700
Rozszerzanie podstawowego typu: string	702



LINQ

15

Przejmij kontrolę nad danymi

To świat przepelniony danymi... lepiej żebyś wiedział, jak w nim żyć. Czasy, gdy mogłeś programować kilka dni, a nawet kilka tygodni, bez konieczności pracy z **ogromem danych**, minęły już bezpowrotnie. Nadeszła epoka, w której **wszystko opiera się na danych**. W rzeczywistości dość często będziesz musiał pracować z danymi, które pochodzą z więcej niż **jednego źródła**... i będą zapisane w różnych formatach. Bazy danych, XML, kolekcje z innych programów... wszystko to jest częścią pracy dobrego programisty C#. W tym miejscu do dzieła wkracza LINQ. To nie tylko sposób na **pobieranie danych** w prosty, intuicyjny sposób. Pozwala on także **grupować i łączyć dane pochodzące z różnych źródeł**.

Łatwy projekt...	708
...ale dane są w różnych miejscach	709
Dzięki LINQ możesz pobrać dane z różnych źródeł	710
Kolekcje .NET są przystosowane do działania z LINQ	711
LINQ ułatwia wykonywanie zapytań	712
LINQ jest prosty, ale Twoje zapytania wcale takie być nie muszą	713
LINQ ma wiele zastosowań	716
LINQ może połączyć Twoje wyniki w grupy	721
Połącz wartości Janka w grupy	722
Użyj Join do połączenia dwóch kolekcji w jednym zapytaniu	725
Janek zaoszczędził mnóstwo szmału	726
Połącz LINQ z bazą danych SQL	728
Użyj join, aby połączyć dane Starbuzz i Papierni Obiektowo	732



Laboratorium C# numer 3

Invaders

Dzięki temu laboratorium oddasz hołd jednej z najbardziej popularnych, czczonych i powielanych ikon w historii gier komputerowych. Nie potrzebuje ona żadnego wprowadzenia. Czas utworzyć grę Invaders.

Dziadek wszystkich gier	736
Można zrobić znacznie więcej...	755



Pozostałości

A

11 najważniejszych rzeczy, które chcieliśmy umieścić w tej książce


Zabawa dopiero się zaczyna! Pokazaliśmy Ci mnóstwo wspaniałych narzędzi do tworzenia naprawdę **potężnych programów** w C#. Nie jest jednak możliwe, abyśmy w tej książce zmieścili **każde narzędzie, technologię i technikę** — nie ma ona po prostu tylu stron. Musieliśmy podjąć *naprawdę przemyślaną decyzję*, co umieścić, a co pominąć. Oto kilka tematów, których nie mogliśmy przedstawić. Pomimo tego, że nie zajęliśmy się nimi, w dalszym ciągu myślimy, że są one **ważne i przydatne**. Należałoby więc chociaż o nich wspomnieć — tak też zrobiliśmy.

1. Podstawy	758
2. Przestrzenie nazw i złożenia	764
3. Użyj BackgroundWorker, by poprawić działanie interfejsu użytkownika	768
4. Klasa Type oraz metoda GetType()	771
5. Równość, IEquatable oraz Equals()	772
6. Stosowanie yield return do tworzenia obiektów umożliwiających iterację	775
7. Refaktoryzacja	778
8. Anonimowe typy i metody oraz wyrażenia lambda	780
9. Serializacja przy użyciu DataContractSerializer	782
10. Zastosowanie LINQ to XML	784
11. Windows Presentation Foundation	786
Czy wiesz, że C# i .NET Framework potrafią...	788


S

Skorowidz

791

 backgroundWorker1

 fileSystemWatcher1

 performanceCounter1

1. Zwiększ wydajność przy pomocy C#

✦ Aplikacje Visual Studio w 10 minut lub mniej ✦

Nie martw się
mamo. Z Visual Studio oraz C#
będziesz mogła programować
tak szybko, że już nigdy więcej
nie przypalisz garnka
z pieczenią.



Czy chcesz tworzyć wspaniałe programy naprawdę szybko? Wraz z C# dostajesz do ręki **potężny język programowania** i wartościowe narzędzie. Dzięki **Visual Studio IDE** do historii przejdą sytuacje, w których musiałeś pisać jakiś nędzny kod, by ponownie zapewnić prawidłowe działanie przycisku. I to nie wszystko. Dodatkowo będziesz mógł skupić się na **faktycznym wykonywaniu swojej pracy**, zamiast starać się zapamiętać, który parametr metody odpowiadał za *nazwę przycisku*, a który za *wyświetlany na nim tekst*. Brzmi zachęcająco? Przewróć zatem stronę i przystąpmy do programowania.

Dlaczego powinieneś uczyć się C#

C# oraz Visual Studio IDE ułatwiają Ci poznanie tajników pisania kodu i, co ważne, pisania go szybko. Kiedy pracujesz z C#, pakiet Visual Studio jest Twoim najlepszym przyjacielem oraz stałym kompanem.

↖ IDE — lub *Visual Studio Integrated Development Environment* — pełni ważną rolę w pracy z C#. To program, który pozwala Ci edytować kod, zarządzać plikami, a także publikować Twoje projekty.

A oto lista czynności, które IDE wykonuje za Ciebie:

Za każdym razem, kiedy zamierzasz rozpocząć pisanie programu lub chociażby umieścić przycisk na formularzu, Twój program potrzebuje całej masy powtarzającego się kodu.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Linq;
namespace A_New_Program
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

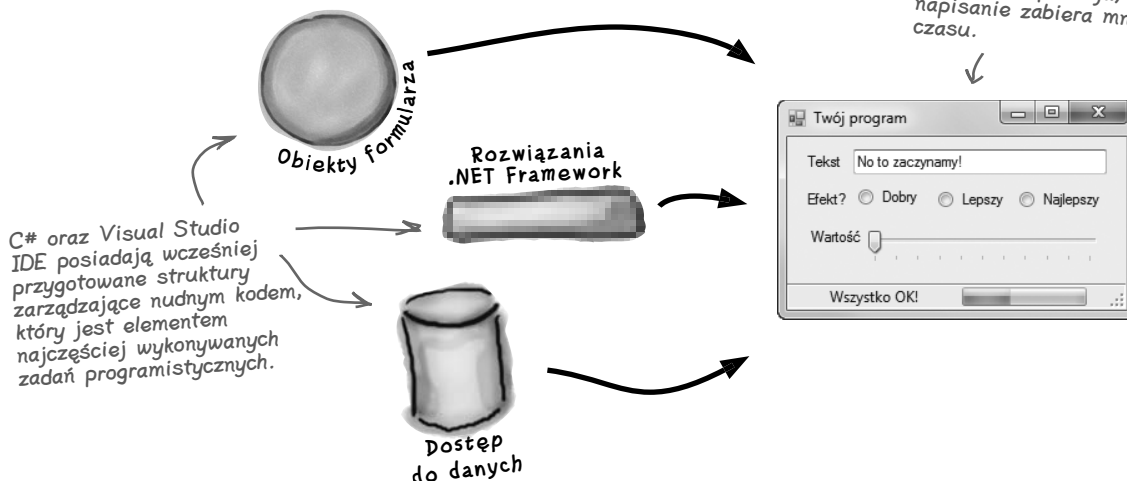
```
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(105, 56);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(75, 23);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    this.button1.UseVisualStyleBackColor = true;
    this.button1.Click += new System.EventHandler(this.button1_Click);
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(292, 267);
    this.Controls.Add(this.button1);
    this.Name = "Form1";
    this.Text = "Form1";
    this.ResumeLayout(false);
}
```

↖ Tyle kodu potrzeba, aby narysować przycisk na formularzu. Dodanie do niego nieco większej ilości elementów wizualnych może spowodować, że wynikowy kod będzie dziesięć razy dłuższy.

Co otrzymujesz razem z Visual Studio oraz C#?

Z językiem C#, przystosowanym do programowania Windows, oraz Visual Studio IDE możesz natychmiast skupić się na tym, co powinien **robić** Twój program.

↓ Wynikiem jest lepiej działająca aplikacja, której napisanie zabiera mniej czasu.



C# oraz Visual Studio ułatwiają wiele czynności

Kiedy używasz C# i Visual Studio, dostajesz wiele wspaniałych możliwości bez żadnego dodatkowego nakładu pracy. Reasumując, uzyskujesz możliwość:

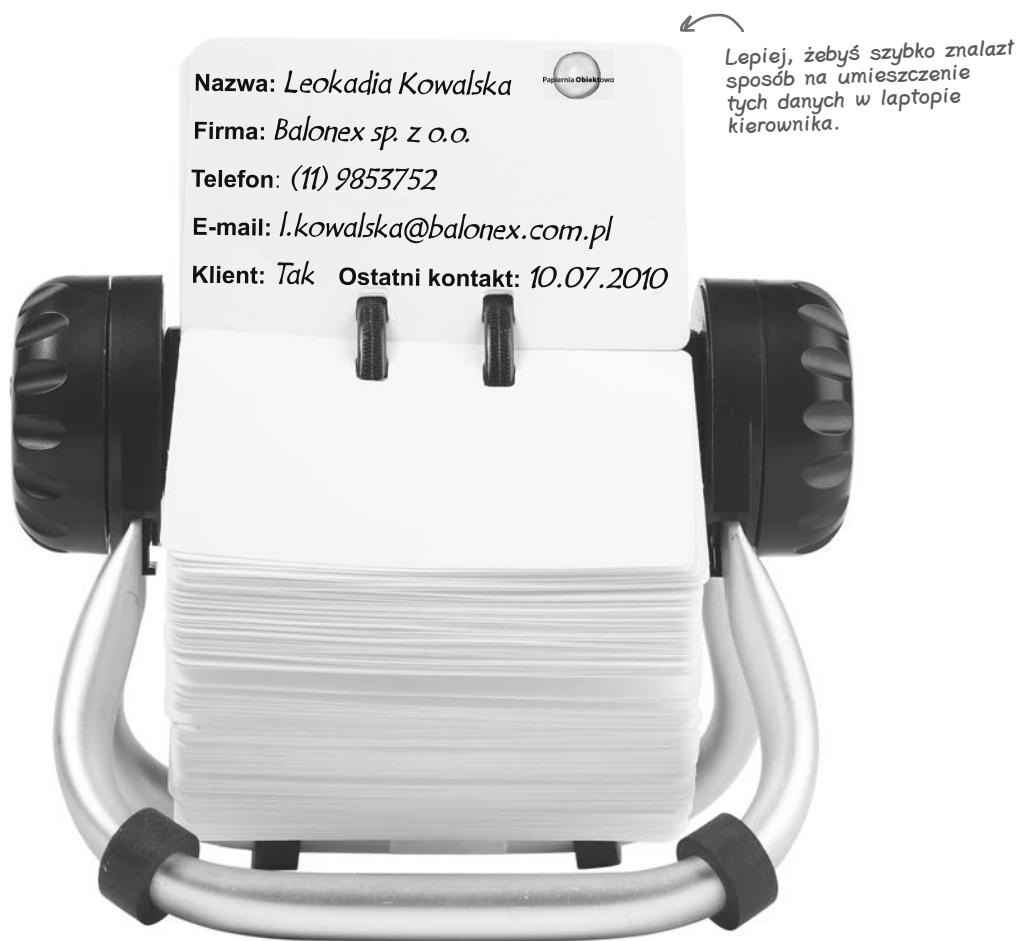
- ❶ **Tworzenia aplikacji SZYBKO.** Tworzenie programów w C# jest niczym robienie zdjęć aparatem cyfrowym. Język jest potężny i łatwy do opanowania, natomiast Visual Studio IDE przejmuje ogromną część pracy i wykonuje ją za Ciebie automatycznie. Możesz zostawić przyziemne sprawy związane z kodowaniem IDE, a samemu skupić się na tym, co Twój kod powinien wykonywać.
- ❷ **Zaprojektowania wspaniale wyglądającego interfejsu użytkownika.** Form Designer w środowisku Visual Studio IDE jest jednym z najprostszych istniejących narzędzi do projektowania. Robi za Ciebie tak wiele, że kreowanie oszałamiających interfejsów staje się jedną z najbardziej satysfakcjonujących czynności podczas tworzenia aplikacji w C#. Możesz budować profesjonalne, w pełni funkcjonalne programy bez niepotrzebnego tracenia wielu godzin na pisanie po raz kolejny od podstaw graficznego interfejsu użytkownika.
- ❸ **Tworzenia i zarządzania bazami danych.** IDE wyposażone jest w prosty interfejs służący do budowania baz danych. Pozwala on na bezproblemową integrację z SQL Server Compact Edition, jak również z kilkoma innymi systemami bazodanowymi.
- ❹ **Skupienia się na rozwiązywaniu PRAWDZIWYCH problemów.** IDE robi za Ciebie wiele, ale w dalszym ciągu to Ty panujesz nad tym, co tworzysz za pomocą C#. IDE pozwala Ci skupić się na Twoim programie, pracy (lub zabawie!) oraz klientach, a do tego zajmuje się całą czarną robotą, taką jak:
 - ◆ śledzenie wszystkich Twoich projektów,
 - ◆ ułatwanie edycji kodu,
 - ◆ kontrolowanie grafiki, dźwięków, ikon oraz innych zasobów w Twoich projektach,
 - ◆ zarządzanie i interakcja z bazami danych.

Oznacza to, że cały ten czas, jaki musiałbyś spędzić, wykonując rutynowe zadania, można przeznaczyć na **tworzenie zabójczych programów**.

← Wkrótce dowiesz się,
co naprawdę mamy na myśli.

Pomóż dyrektorowi naczelnemu zrezygnować z papieru

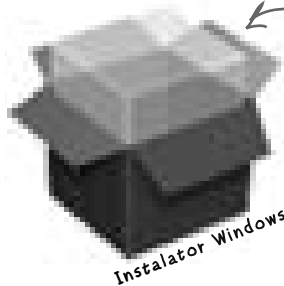
Firma papiernicza Papiernia Obiektowo właśnie zatrudniła nowego dyrektora naczelnego. Uwielbia on wycieczki piesze, kawę, przyrodę... i podjął decyzję, że będzie przyczyniał się do ratowania lasów. Ma potrzebę zostania kierownikiem „elektronicznym”. Na pierwszy ogień pójdzie lista jego kontaktów. W najbliższy weekend wybiera się na narty do Aspen i liczy na to, że do jego powrotu powstanie nowy program z książką adresową. W przeciwnym razie... no cóż... nie tylko stary dyrektor naczelny będzie szukał pracy.



Sprawdź potrzeby Twoich użytkowników, zanim zaczniesz tworzyć program

Zanim zaczniesz pisać książkę adresową — lub *jakąkolwiek* inną aplikację — musimy zatrzymać się na minutę i pomyśleć o tym, kto będzie jej używał i czego od niej oczekuje.

- 1 Dyrektor naczelny chce uruchamiać program w pracy oraz na swoim laptopie. Potrzebuje zatem programu instalacyjnego, aby mieć pewność, że wszystkie niezbędne pliki znajdują się na każdym komputerze.



Dyrektor naczelny chce uruchamiać książkę adresową na komputerze stacjonarnym i laptopie, zatem program instalacyjny jest niezbędny.

- 2 Dział sprzedaży firmy papierniczej Papiernia Obiektowo także chce mieć dostęp do książki adresowej. Chce on używać danych do budowania list kontaktów i w ten sposób zwiększyć sprzedaż papieru.

Dyrektor naczelny dochodzi do wniosku, że baza danych będzie najlepszym rozwiązaniem. W ten sposób każdy w firmie będzie miał dostęp do tych informacji, a on będzie musiał zarządzać tylko jedną kopią swoich kontaktów.

Wiemy już, że Visual Studio ułatwia pracę z bazami danych. Przechowywanie informacji w takiej formie umożliwia jednoczesny dostęp do nich dyrektorowi naczelnemu oraz działowi sprzedaży, mimo że fizycznie istnieje tylko jedna kopia danych.

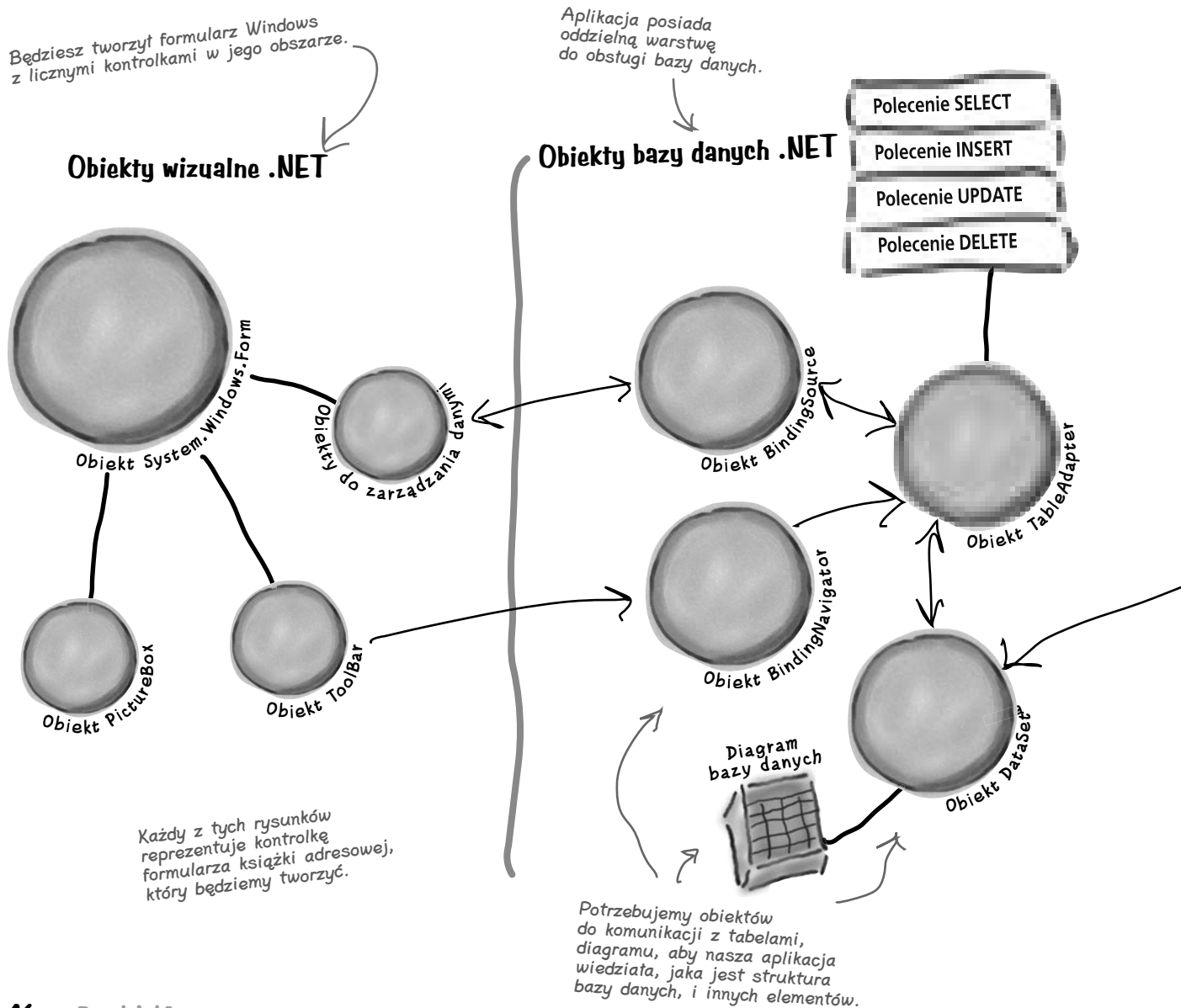


Zanim zaczniesz tworzyć kod, pomyśl o swoich użytkownikach oraz ich potrzebach; oni na pewno to docenią i będą zadowoleni z finalnego produktu, kiedy go wreszcie skończysz.

Oto program, który zamierzasz stworzyć

Potrzebujesz aplikacji z graficznym interfejsem użytkownika, obiektów niezbędnych do komunikacji z bazą danych, właściwej bazy danych oraz programu instalacyjnego. Brzmi to jak wyrok i zapowiada konieczność wygospodarowania ogromnej ilości czasu, ale zbudujesz to wszystko po przeanalizowaniu następujących kilku stron.

A oto struktura programu, który zamierzamy stworzyć:



Wszystkie dane przechowywane są w tabeli bazy danych SQL Server Compact.

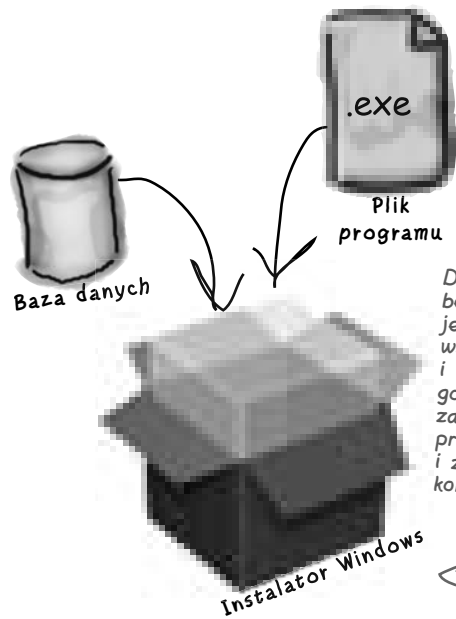
Pamięć bazy danych



To jest prawdziwa baza danych, którą możemy utworzyć i którą można zarządzać za pomocą Visual Studio.

Program po utworzeniu umieszczony jest w pliku instalatora Windows.

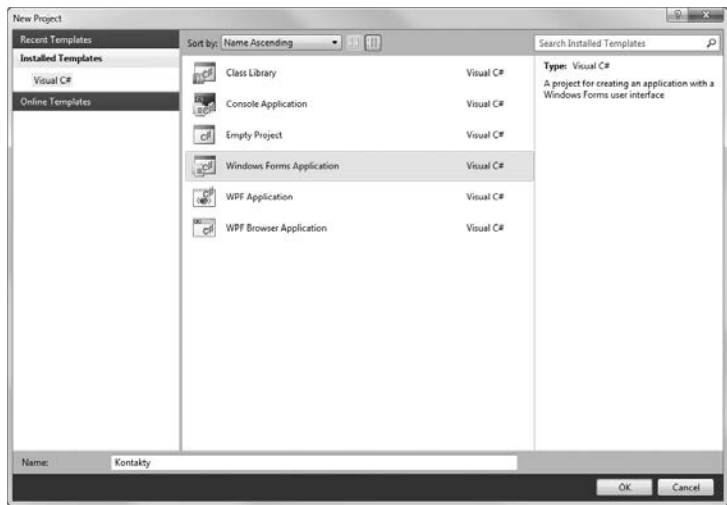
Paczka instalatora



Dział sprzedaży będzie musiał jedynie wskazać plik i kliknąć go, aby zainstalować program i z niego korzystać.

Co robisz w Visual Studio

Przystąp więc do dzieła i uruchom Visual Studio, jeżeli jeszcze tego nie zrobiłeś. Pomiń stronę startową i wybierz *New Project* z menu *File*. Nazwij swój projekt „Kontakty” i naciśnij OK. Do wyboru masz kilka różnych typów projektów. Wybierz *Windows Forms Application*, a jako nazwę swojego nowego projektu wpisz „Kontakty”.



Wygląd IDE może się nieco różnić od Twojego.

Tak wygląda okno *New Project* w Visual Studio 2010 Express Edition. Jeśli używasz wersji *Professional* lub *Team Foundation*, może ono wyglądać nieco inaczej. Ale nie przejmuj się — wszystko będzie działało dokładnie tak samo.

Co Visual Studio robi za Ciebie

Gdy tylko zapiszesz nowy projekt, IDE utworzy pliki *Form1.cs*, *Form1.Designer.cs* oraz *Program.cs*. Elementy dodawane są do okna *Solution Explorer*. Domyślnie pliki zapisywane są także w folderze *Moje dokumenty\Visual Studio 2010\Projects\Kontakty*.

Upewnij się, że zaraz po utworzeniu projektu zapiszesz go, wybierając *Save All* z menu *File* — spowoduje to zapisanie do katalogu wszystkich jego plików. Jeżeli wybierzesz *Save*, zapisany zostanie tylko plik, nad którym aktualnie pracujesz.

Ten plik zawiera kod C# definiujący zachowanie formularza.

Ten zawiera kod uruchamiający program i wyświetlający formularz.

Kod, który definiuje wygląd formularza oraz jego obiektów, znajduje się tutaj.



Form1.cs



Program.cs



Form1.Designer.cs

Visual Studio tworzy wszystkie trzy pliki automatycznie.



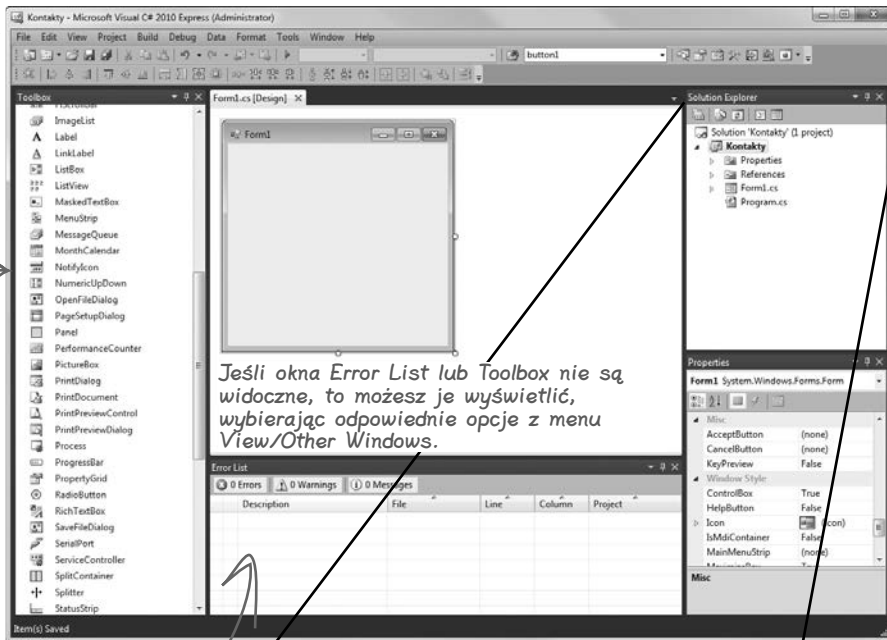
Zaostrz ołówek

Poniżej pokazany jest wygląd ekranu, jaki prawdopodobnie możesz w tej chwili zaobserwować. Powinieneś odgadnąć, do czego służą poszczególne okna, bazując na wiedzy, którą na tym etapie posiadasz. Upewnij się, że okna *Toolbox* oraz *Error List* są widoczne, wybierając odpowiednie opcje z menu *View/Other Windows*. W każdym z pustych pól wstaw komentarz wyjaśniający przeznaczenie danego obszaru IDE. Na początek uzupełniliśmy jedno pole.

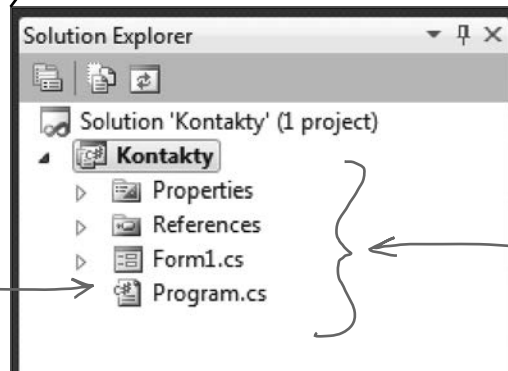
Ten pasek narzędzi posiada przyciski związane z tym, co jest aktualnie wykonywane za pomocą IDE.

Jeżeli Twoje IDE nie wygląda dokładnie tak jak na tym obrazku, możesz wybrać *Reset Window Layout* z menu *Window*.

Powiększyliśmy to okno, abyś miał więcej miejsca.



Jeśli okna *Error List* lub *Toolbox* nie są widoczne, to możesz je wyświetlić, wybierając odpowiednie opcje z menu *View/Other Windows*.



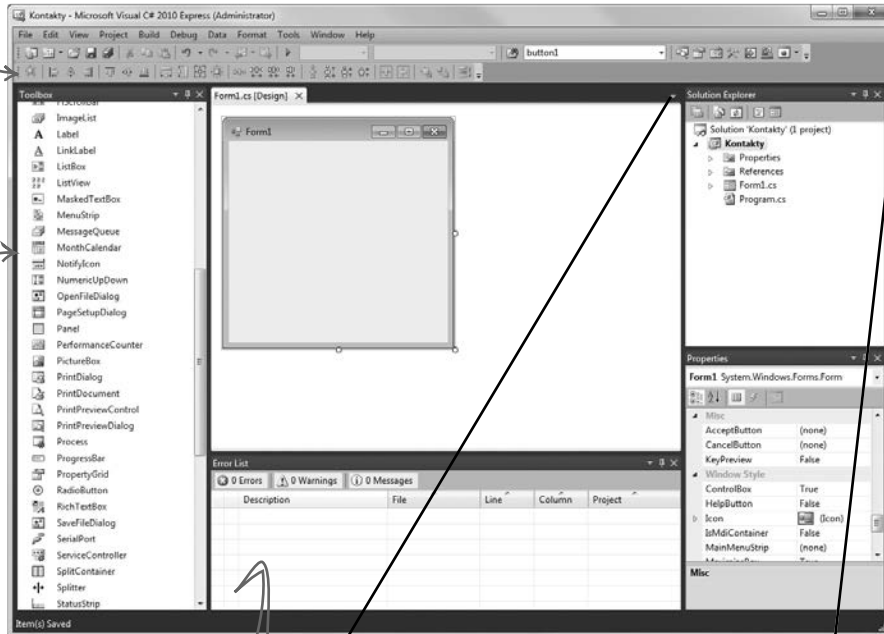
Zaostrz ołówek **Rozwiązanie**



Uzupełniliśmy komentarze dotyczące różnych obszarów Visual Studio C# IDE. W poszczególnych miejscach możesz mieć napisane inne rzeczy, ale powinieneś znać przeznaczenie każdego okna i każdej sekcji.

Ten pasek narzędzi posiada przyciski związane z tym, co jest aktualnie wykonywane za pomocą IDE.

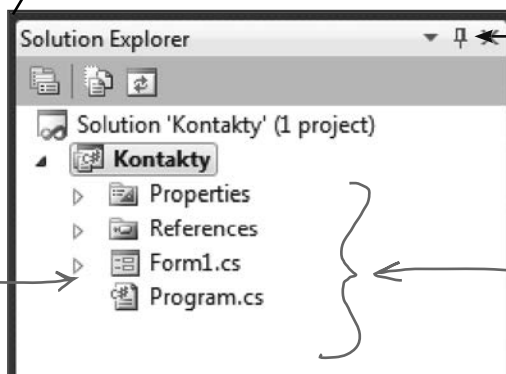
To jest okno z kontrolkami. Zawiera zestaw kontrol, które możesz przeciągać wprost na formularz.



To okno pokazuje wszystkie właściwości aktualnie wybranej kontrolki formularza.

Jeśli w kodzie pojawią się jakieś błędy, zostaną wyświetlone w oknie Error List (lista błędów). Ten dolny panel pokazuje wiele informacji diagnostycznych dotyczących tworzonego programu.

W oknie Solution Explorer są wyświetlone pliki Form1.cs oraz Program.cs, które zostały utworzone dla Ciebie przez IDE podczas dodawania nowego projektu.



Czy widzisz tę niewielką ikonkę pineski? Klikając ją, możesz włączać lub wyłączać automatyczne ukrywanie okna. W przypadku okna Toolbox automatyczne ukrywanie jest domyślnie włączone.

Możesz przetaczać się pomiędzy plikami za pomocą okna Solution Explorer w IDE

P: Skoro IDE pisze cały ten kod za mnie, to czy nauka C# sprowadza się do nauki IDE?

U: Nie. IDE jest wspianiałe w automatycznym generowaniu dla Ciebie części kodu, ale więcej nie może zrobić. Jest wiele rzeczy, w których jest naprawdę dobre, takich jak wstępna konfiguracja lub automatyczna zmiana właściwości kontrolki na formularzu. Istnieje jednak znacznie trudniejsza część programowania, czyli troska o to, co powiniem robić program, i zmuszenie go do posłuszeństwa, a tego żadne IDE za Ciebie nie zrobi. Pomimo tego, że Visual Studio IDE jest jednym z najbardziej zaawansowanych spośród istniejących środowisk, może ono dojść tylko do tego etapu. To Ty, nie żadne IDE, jesteś odpowiedzialny za pisanie kodu, który wykonuje właściwą pracę.

P: Utworzyłem nowy projekt w Visual Studio, ale kiedy sprawdziłem podkatalog Projects w folderze Moje dokumenty, nie znalazłem go tam. Co się stało?

U: Kiedy zaczynasz tworzyć nowy projekt w Visual Studio 2010, IDE umieszcza go w katalogu *Ustawienia lokalne \ Dane aplikacji \ Temporary Projects*. Gdy zapisujesz go po raz pierwszy, jesteś pytany o nazwę pliku, a po zaakceptowaniu wyboru projekt zostaje zapisany w katalogu *Moje dokumenty \ Visual Studio 2010 \ Projects*. Próbując otworzyć nowy projekt lub zamknąć tymczasowy, zostaniesz poproszony o wybór jednej z dwóch opcji — zapisania lub odrzucenia zmian. (Uwaga: inne wersje Visual Studio nie używają tego tymczasowego katalogu do przechowywania projektów — tworzą je od razu w katalogu *Projects!*)

P: Co wtedy, gdy IDE utworzy kod, którego nie chciałem?

U: Możesz go zmienić. IDE jest przystosowane do tworzenia kodu na podstawie tego, w jaki sposób elementy zostały przeciągnięte lub dodane do aplikacji. Czasami nie jest to sposób, jaki byłby przez Ciebie pożądanym. Wszystko, co IDE robi dla Ciebie — każda linijka kodu, którą tworzy, każdy plik, który dodaje — może być zmieniane zarówno ręcznie, poprzez bezpośrednią edycję plików, jak i przy użyciu łatwego w obsłudze interfejsu środowiska.

P: Czy wszystko będzie działało dobrze, jeżeli pobrałem i zainstalowałem Visual Studio Express? Czy nie powinienem używać jednej z płatnych wersji Visual Studio, aby zrobić wszystko, co jest tu opisane?

U: Nie ma w tej książce niczego, czego nie dałoby się zrobić za pomocą darmowej wersji Visual Studio (którą można pobrać ze strony Microsoftu). Główne różnice pomiędzy edycją Express a pozostałymi (Professional i Team Foundation) nie uniemożliwiają pisania w pełni funkcjonalnych i kompletnych aplikacji w C#.

P: Czy mogę zmienić nazwy plików, które IDE dla mnie generuje?

U: Oczywiście. Podczas tworzenia nowego projektu IDE generuje domyślny formularz o nazwie *Form1* — w jego skład wchodzi trzy pliki: *Form1.cs*, *Form1.Designer.cs* oraz *Form1.resx*. Możesz jednak skorzystać z okna *Solution Explorer*, by nadać tym plikom dowolne inne nazwy. Domyślnie odpowiadają one nazwie formularza.

Jeśli zmienisz nazwy plików, to w oknie *Properties* będziesz miał okazję przekonać się, że formularz wciąż nosi nazwę *Form1*. Jego nazwę możesz zmienić w wierszu „(Name)” w oknie *Properties*. Jeśli to zrobisz, nazwa pliku nie ulegnie zmianie. C# nie zwraca uwagi na nazwy plików i formularzy (ani jakichkolwiek innych elementów projektu), choć istnieje kilka reguł, którymi należy się kierować podczas ich określania. Niemniej jednak dobieranie odpowiednich nazw może ułatwić Ci pracę. Póki co nie przejmuj się nimi — w dalszej części książki zamieścimy znacznie więcej informacji dotyczących dobierania nazw dla różnych elementów programów.

P: Przyglądam się właśnie mojemu IDE i dochodzę do wniosku, że mój ekran wygląda nieco inaczej niż Twój. Brakuje niektórych okien, inne są poprzestawiane. Co się dzieje?

U: Jeżeli wybierzesz polecenie *Reset Window Layout* z menu *Window*, to IDE odtworzy oryginalny wygląd i położenie okien. Następnie możesz skorzystać z opcji *View/Other Windows*, by upodobnić swoje okno Visual Studio do tych pokazanych w tym rozdziale.

Visual Studio generuje kod, który może być użyty jako podstawa do dalszego pisania aplikacji.

Upewnienie się, że aplikacja robi dokładnie to, do czego została stworzona, należy wyłącznie do Ciebie.

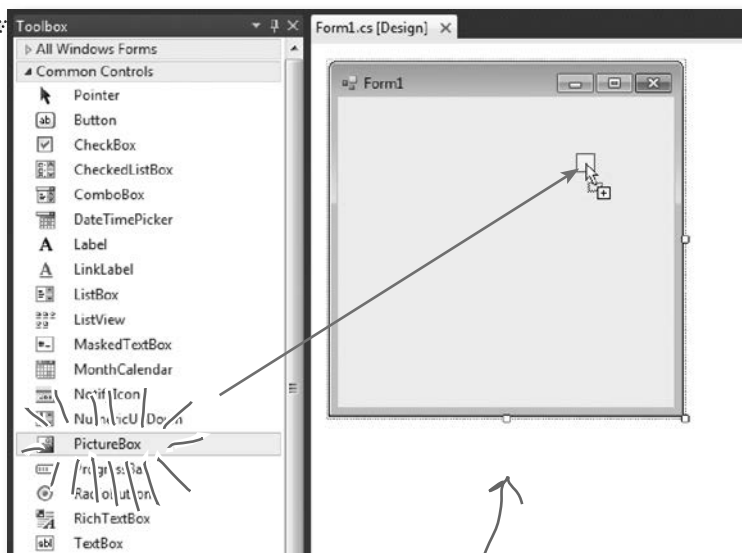
Stwórz interfejs użytkownika

Dodawanie kontrolki i nadawanie blasku interfejsowi użytkownika jest bardzo proste i sprowadza się do przeciągania i upuszczania elementów w Visual Studio IDE. Dodajmy zatem logo do formularza.

1 Użyj kontrolki PictureBox, aby dodać obrazek.

Kliknij kontrolkę *PictureBox* w oknie *Toolbox*, a następnie przeciągnij ją na formularz. W tle IDE doda jej kod do pliku *Form1.Designer.cs*.

Jeżeli nie widzisz okna *Toolbox*, spróbuj najechać wskaźnikiem myszy na słowo *Toolbox*, które znajduje się w lewym górnym rogu IDE. Jeśli go tam nie ma, wybierz element *Toolbox* z menu *View*, aby pojawiło się na ekranie.



Za każdym razem, gdy wprowadzasz zmiany we właściwościach kontrolki formularza, kod w pliku *Form1.Designer.cs* jest zmieniany przez IDE.



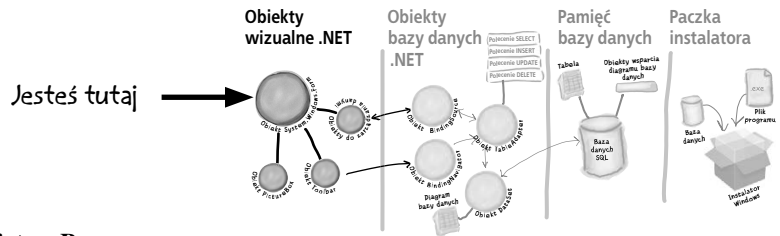
Form1.Designer.cs



Spokojnie

Nie przejmuj się, jeżeli nie jesteś profesjonalistą w projektowaniu graficznych interfejsów użytkownika.

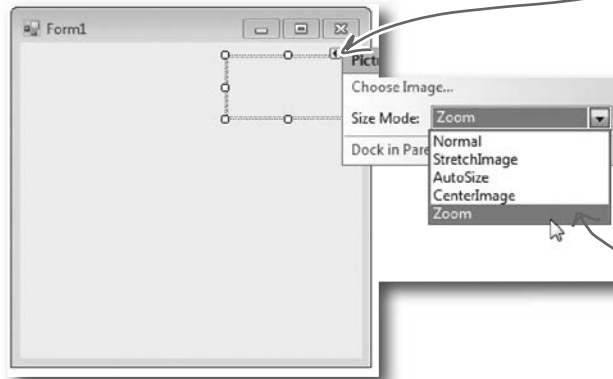
W swoim czasie powiemy znacznie więcej o tworzeniu dobrych interfejsów użytkownika. Na tym etapie wystarczy, że wstawisz logo oraz inne kontrolki do formularza i zadbasz o jego prawidłowe **zachowanie**. O stylowy wygląd zatroszczymy się później.



2 Ustaw tryb Zoom dla obiektu PictureBox.

Każda kontrolka na Twoim formularzu posiada właściwości, które możesz ustawić. Aby dostać się do nich, kliknij małą czarną strzałkę widoczną na kontrolce. Zmień właściwość *Size Mode* obiektu *PictureBox* na „Zoom”, a zobaczysz, jak to działa.

Wartość właściwości *Size Mode* można także określić w oknie *Properties IDE*. Ta niewielka czarna strzałka ma jedynie zapewnić łatwiejszy dostęp do najczęściej używanych właściwości kontrolki.



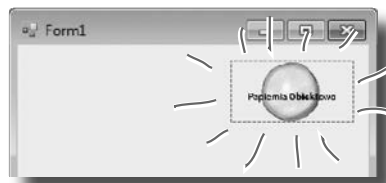
Aby uzyskać dostęp do właściwości kontrolki, kliknij tę małą czarną strzałkę.

Wybierz *Zoom*, aby ramka *PictureBox* zmieniła swój rozmiar w celu dopasowania się do obrazka, który w niej umieścisz.

Następnie kliknij opcję *Choose Image*, aby wyświetlić okno dialogowe *Select Resource* i za jego pomocą zaimportować wybrany obrazek z lokalnego dysku.

3 Pobierz logo firmy Papiernia Obiektowo.

Pobierz logo Papierni Obiektowo z serwera ftp (<ftp://ftp.helion.pl/przyklady/cshrug.zip>) i zapisz je na twardym dysku. Następnie wybierz strzałkę właściwości kontrolki *PictureBox* i kliknij *Choose Image*. Na ekranie pojawi się okno dialogowe *Select Resource*. Zaznacz przycisk opcji *Local Resource*, aby uaktywnić przycisk *Import*; kliknij ten przycisk a następnie odszukaj na dysku obrazek, którego chcesz użyć. I to już wszystko.

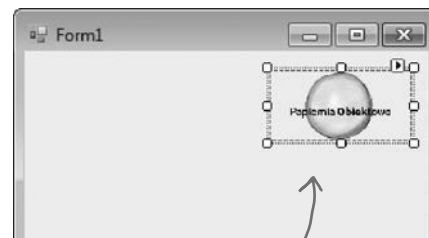


Tutaj znajduje się logo Papierni Obiektowo oraz kontrolka *PictureBox*, która dopasowała swój rozmiar do jego wielkości.

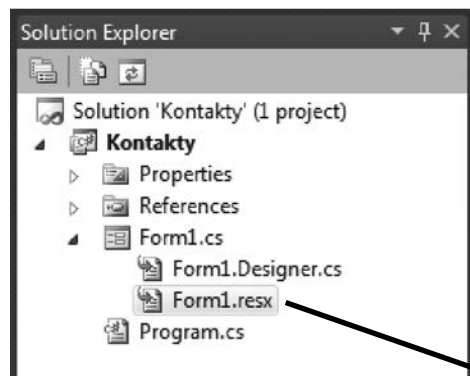
Za kulisami Visual Studio

Za każdym razem, gdy zmienisz coś w Visual Studio, IDE *pisze za Ciebie kod*. Kiedy utworzyłeś logo i zdecydowałeś, żeby Visual Studio używało pobranego obrazka, stworzyło ono zasób i skojarzyło go z Twoją aplikacją. **Zasoby** to każdy plik graficzny, dźwiękowy, ikona, a także każdy inny rodzaj danych wrzucanych do Twojego programu. Plik graficzny staje się wówczas integralną częścią aplikacji, aby podczas jej instalacji na innym komputerze został tam zapisany razem z innymi składnikami i aby *PictureBox* mógł go używać.

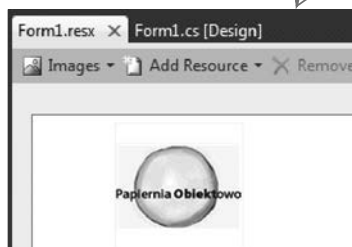
Kiedy przeciągnąłeś kontrolkę *PictureBox* na formularz, IDE automatycznie utworzyło plik *Form1.resx*, aby przechowywać zasoby i trzymać je razem w projekcie. Kliknij dwukrotnie ten plik, a zobaczysz niedawno zaimportowany obrazek.



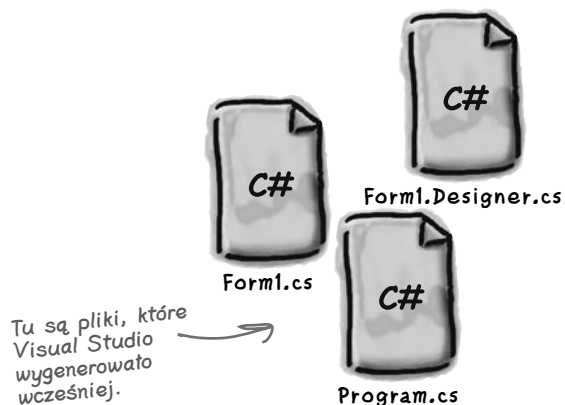
Ten obrazek jest teraz w zasobach aplikacji listy kontaktowej.



Przejdź do okna *Solution Explorer* i kliknij niewielki trójkącik wyświetlony z lewej strony pliku *Form1.cs*, aby rozwinąć dodatkowe opcje (oczywiście, o ile nie są jeszcze widoczne). W ten sposób w oknie powinny pojawić się kolejne dwa pliki: *Form1.Designer.cs* oraz *Form1.resx*. Aby wyświetlić zaimportowane wcześniej logo, dwukrotnie kliknij ten drugi plik, a następnie kliknij na strzałce umieszczonej obok opcji *Strings* i z rozwijalnej listy wybierz opcję *Image* (możesz także nacisnąć *Ctrl+2*). To właśnie ten plik łączy logo firmy z kontrolką *PictureBox*, a IDE automatycznie dodało kod, który obsługuje to połączenie.



Jeśli w oknie dialogowym *Select Resource* klikniesz drugi przycisk *Import* (w ramach czynności opisanych na poprzedniej stronie), to obrazek pojawi się w katalogu *Resources* (widocznym w oknie *Solution Explorer*). Nie przejmuj się tym — wróć ponownie do okna *Select Resource* i jeszcze raz zaimportuj logo do zasobów programu — w ten sposób pojawi się ono we właściwym miejscu.



Dodaj coś do automatycznie wygenerowanego kodu

IDE tworzy dużą część kodu za Ciebie, ale zawsze będziesz chciał go przeglądać i coś do niego dodawać. Ustawmy zatem logo w ten sposób, aby wyświetlało okienko informacyjne, gdy użytkownik uruchomi program i je kliknie.

Podczas edycji formularza w IDE dwukrotne kliknięcie któregośkolwiek z jego elementów spowoduje automatyczne dodanie kodu do projektu. Upewnij się, że IDE wyświetla formularz, a następnie dwukrotnie kliknij kontrolkę *PictureBox*. W rezultacie IDE doda do projektu kod, który będzie wykonywany za każdym razem, gdy użytkownik kliknie logo. Powinieneś zobaczyć fragment pojawiającego się kodu podobny do tego:



Gdy dwukrotnie kliknąłeś kontrolkę *PictureBox*, IDE utworzyło tę metodę. Będzie ona wykonywana za każdym razem, gdy użytkownik kliknie logo podczas działania aplikacji.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void pictureBox1_Click (object sender, EventArgs e)
    {
        MessageBox.Show("Lista kontaktów v1.0.\nAutor: Twoje imię", "O programie");
    }
}
```

Nazwa tej metody doskonale określa, kiedy jest ona wywoływana: gdy ktoś kliknie tę kontrolkę *PictureBox*.

Gdy dwukrotnie klikniesz w kontrolkę *PictureBox*, otworzy ona ten kod z kursorem ustawionym w tym miejscu. Ignoruj wszystkie okna, które pojawiają się podczas wpisywania. IDE stara się pomóc, ale na razie tej pomocy nie potrzebujemy.

Wpisz tę linijkę kodu. Powoduje ona wyświetlenie okienka dialogowego prezentującego podany tekst. Okno będzie miało tytuł „O programie”.

Jeśli już wpisałeś ten fragment kodu, zapisz go, używając ikony Save z paska narzędzi IDE lub poprzez wybór opcji Save z menu File. Przyzwyczajaj się do częstego zapisywania rezultatów swojej pracy przy pomocy opcji Save All.

Nie istnieją
głupie pytania

P: Co to jest metoda?

O: Metoda to po prostu *nazwany fragment kodu*. Powiemy sobie na ten temat znacznie więcej w rozdziale 2.

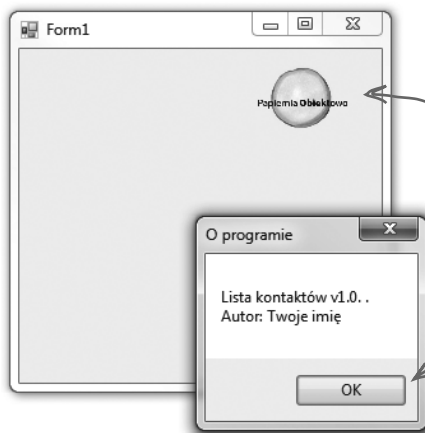
P: Co robi jakiś `\n` w kodzie?

O: To symbol łamania wierszy tekstu. W ten sposób mówimy C#, aby umieścić fragment "Lista kontaktów v1.0." w jednym wierszu, a pozostałą część: "Autor", w kolejnym.

Już możesz uruchomić aplikację

Naciśnij F5 na klawiaturze lub kliknij przycisk z zieloną strzałką (▶) na pasku narzędzi, aby sprawdzić, co udało Ci się do tego momentu napisać. (Nazywa się to „debugowaniem”, czyli uruchamianiem programu za pośrednictwem IDE). Możesz zatrzymać debugowanie, wybierając *Stop Debugging* z menu *Debug* lub klikając w ten przycisk na pasku narzędzi: (■).

← Wszystkie trzy przyciski działają — nie musisz pisać w tym celu żadnego kodu.

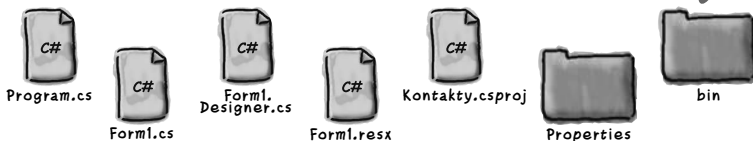


Kliknięcie w logo Papiernia Obiektowa powoduje pokazanie się okna dialogowego, które właśnie napisaliśmy.

Gdzie są moje pliki?

Kiedy uruchamiasz swój program, Visual Studio kopiuje wszystkie pliki do katalogu *Moje Dokumenty\Visual Studio 2010\Projects\Kontakty\Kontakty\bin\debug*. Możesz nawet przejść do niego i uruchomić poprzez dwukrotne kliknięcie pliku *.exe* program, który został utworzony przez IDE.

C# zamienia Twój program na plik, który możesz uruchomić, zwany plikiem wykonywalnym. Znajdziesz go tutaj, w folderze debug.



To nie jest błąd. Istnieją dwa poziomy katalogów. Wewnętrzny folder posiada właściwe pliki z kodem C#.

— Nie istnieją grupie pytania —

P: W moim IDE zielona strzałka jest oznaczona jako *Debug*. Czy to jakiś problem?

O: Nie. Debugowanie, przynajmniej w naszym przypadku, to uruchamianie aplikacji poprzez IDE. Powiemy sobie więcej na jego temat w dalszej części książki. Na chwilę obecną będzie to po prostu sposób na uruchamianie programów.

P: Nie widzę żadnego przycisku *Stop Debugging* na moim pasku narzędzi. O co chodzi?

O: Przycisk *Stop Debugging* znajduje się na specjalnym pasku narzędzi, który staje się widoczny dopiero po uruchomieniu programu. Uruchom aplikację jeszcze raz i sprawdź, czy pasek się pojawił.

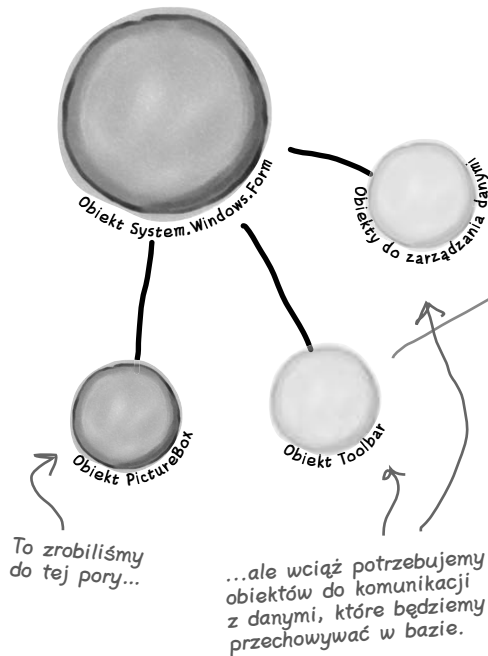
Co zrobiliśmy do tej pory

Stworzyliśmy formularz i wstawiliśmy kontrolkę *PictureBox*, która w momencie kliknięcia wyświetla okienko dialogowe z komunikatem. Kolejnym zadaniem będzie dodanie wszystkich innych pól karty, takich jak nazwa kontaktu i numer telefonu.

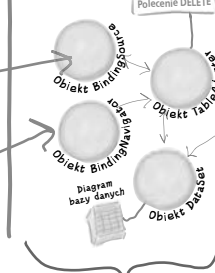
Zdecydowaliśmy, że te informacje będziemy przechowywać w bazie danych. Visual Studio może zarządzać bezpośrednim połączeniem z taką bazą danych, dzięki czemu nie będziemy potrzebowali zaśmiecać kodu licznymi odwołaniami do niej (co jest niewskazane). Jednak aby to wszystko prawidłowo działało, musimy stworzyć naszą bazę w taki sposób, by kontrolki mogły się do niej odwoływać. Zamierzamy więc przenieść się z obszaru obiektów wizualnych .NET bezpośrednio do sekcji bazy danych.



Obiekty wizualne .NET

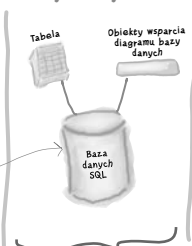


Obiekty bazy danych .NET



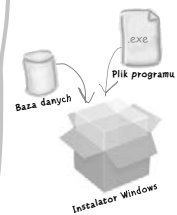
Ten etap ma na celu połączenie formularza z bazą danych, na co nie jesteśmy na razie gotowi. Po prostu nie posiadamy jeszcze bazy danych.

Pamięć bazy danych



Musimy więc skupić się na tym kroku: stworzeniu bazy danych i umieszczeniu w niej pewnych informacji.

Paczka instalatora



Visual Studio może wygenerować kod pozwalający na połączenie formularza i bazy danych, ale ZANIM to będzie możliwe, musimy taką bazę stworzyć.

Potrzebujemy bazy danych do przechowywania naszych informacji

Zanim dodamy do formularza resztę pól, musimy stworzyć bazę, z której będzie on korzystał. IDE może wygenerować znaczną część kodu odpowiedzialnego za połączenie formularza i danych, ale musimy wcześniej zdefiniować samą bazę.

← Upewnij się, że skończyłeś debugować, zanim przejdziesz dalej.

1 Dodaj nową bazę danych SQL do projektu.

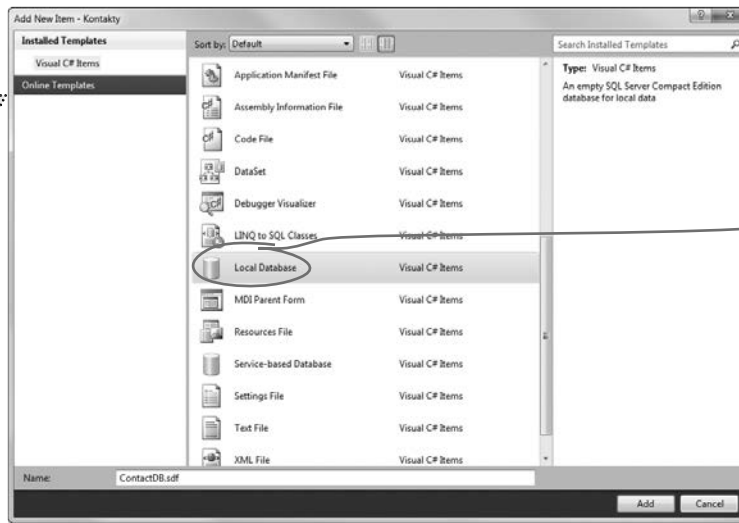
W oknie *Solution Explorer* kliknij prawym przyciskiem myszy projekt *Kontakty* i wybierz *Add*, a następnie *New Item*. Kliknij ikonę *SQL Database*, a jako nazwę wpisz *ContactDB.sdf*.

← Ten plik jest naszą nową bazą danych.



ContactDB.sdf

Wybierz opcję *Local Database*, by utworzyć plik SQL Server Compact Edition, który będzie zawierał całą Twoją bazę danych. Nadaj jej nazwę *ContactDB.sdf*.



Opcja *Local Database* oznacza w rzeczywistości plik bazy danych SQL Server Compact Edition, który zazwyczaj ma rozszerzenie *SDF*. Daje on możliwość łatwego dodania bazy danych do projektu.

2 W oknie dialogowym *Add New Item* kliknij przycisk *Add*.

3 Zamknij okno *Data Source Configuration Wizard*.

Na tym etapie chcemy pominąć konfigurację źródła danych, więc kliknij przycisk *Cancel*. Wrócimy tutaj, jak tylko ustalimy strukturę naszej bazy danych.

4 Oglądnij bazę danych w oknie *Solution Explorer*.

Przejdź do okna *Solution Explorer*, a zobaczysz, że baza *ContactDB* została dodana do listy plików. Kliknij dwukrotnie *ContactDB.sdf* i popatrz na lewą stronę ekranu. Toolbox został zastąpiony przez *Database Explorer*.



Uwaga!

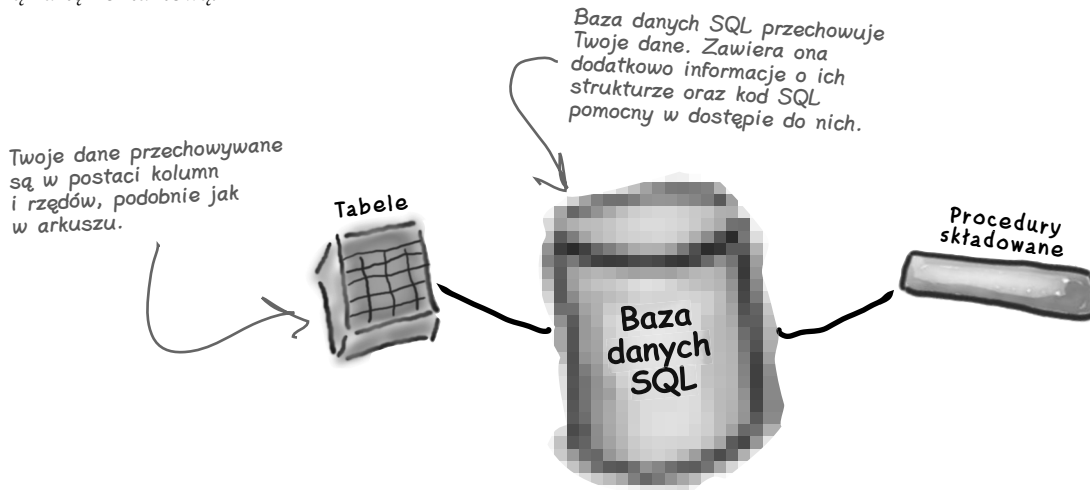
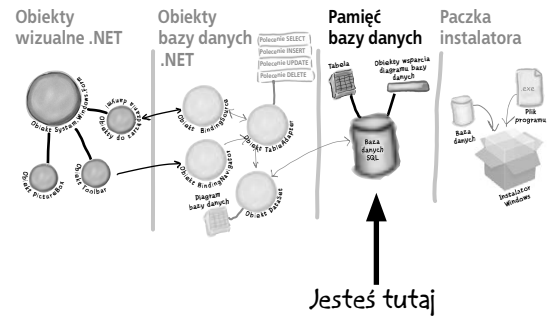
Jeżeli nie używasz edycji Express, możesz zobaczyć *Server Explorer* zamiast okna *Database Explorer*.

Visual Studio 2010 Professional oraz Team Foundation nie posiadają okna *Database Explorer*. Zamiast tego mają okno *Server Explorer*, które potrafi zrobić dokładnie to samo, a oprócz tego umożliwia przeglądanie danych w sieci.

IDE utworzyło bazę danych

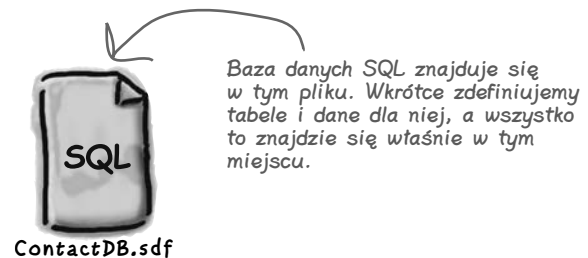
Kiedy nakazałeś IDE dodać nową bazę SQL do projektu, IDE ją utworzyło. **Baza danych SQL** to system, który przechowuje dane w sposób zorganizowany i relacyjny. Dodatkowo IDE udostępnia wszystkie narzędzia niezbędne do pracy z samą bazą danych oraz przechowywanymi w niej informacjami.

Dane w bazie SQL przechowywane są w tabelach. Na bieżące potrzeby wystarczy wyobrazić sobie, że baza to rodzaj arkusza kalkulacyjnego. Organizuje on dane w ramach wierszy i kolumn. Kolumny reprezentują ich kategorie, takie jak nazwa kontaktu i numer telefonu, natomiast każdy rząd reprezentuje jedną kartę kontaktową.



SQL jest swoim własnym językiem

SQL to skrót od **Structured Query Language**. Oznacza język programowania pozwalający na dostęp do danych zgromadzonych w bazie. Posiada on swoją własną składnię, słowa kluczowe i strukturę. Kod SQL przyjmuje postać **instrukcji** i **zapytań**, które umożliwiają dostęp do danych i ich pobieranie. Baza danych może posiadać tak zwane **procedury składowane** będące zestawem instrukcji i zapytań SQL przechowywanych w niej i gotowych do uruchomienia w każdej chwili. IDE generuje instrukcje SQL i procedury składowane automatycznie, aby możliwy był dostęp do danych z poziomu programu.



Baza danych SQL znajduje się w tym pliku. Wkrótce zdefiniujemy table i dane dla niej, a wszystko to znajdzie się właśnie w tym miejscu.

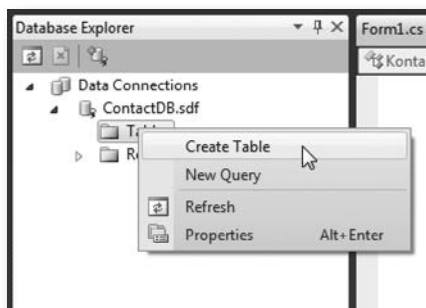
[Notka działu marketingu: Czy możemy wstawić tutaj reklamę książki „SQL. Rusz głową!?”]

Tworzenie tabeli dla listy kontaktowej

Mamy już bazę danych. Potrzebujemy jeszcze zgromadzić w niej informacje, ale muszą się one mieścić w tabelach, czyli strukturach bazy używanych do przechowywania poszczególnych bitów danych. Dla celów naszej aplikacji **stwórzmy zatem tabelę „People”**, aby w niej przechowywać wszystkie informacje kontaktowe.

1 Dodaj tabelę do bazy ContactDB.

Kliknij prawym przyciskiem myszy w element *Tables* w oknie *Database Explorer* i wybierz *Create Table*. Spowoduje to otwarcie okna, w którym możesz zdefiniować nazwy kolumn tworzonej tabeli.



W tej chwili potrzebujemy nowych kolumn w naszej bazie. W pierwszej kolejności do tabeli dodajmy kolumnę *ContactID*, tak aby każdy rekord posiadał swój własny, unikalny identyfikator.

2 Dodaj kolumnę ContactID do tabeli People.

Wpisz „ContactID” w polu Column Name i wybierz *Int* z listy rozwijalnej *Data Type*. Upewnij się, że na liście Allow Nulls została wybrana opcja *No*.

Na koniec utworzymy klucz główny dla naszej tabeli. Podświetl kolumnę *ContactID*, którą właśnie utworzyłeś, i naciśnij przycisk *Primary Key*. Spowoduje to, że każdy wpis w tym polu traktowany będzie jako unikalny klucz główny.



↑ Dodaj nową kolumnę typu *int* o nazwie *ContactID*. Upewnij się, że w polu Allow Nulls została wybrana opcja *No*, a w polach Unique oraz Primary Key opcja *Yes*.

Nie, istnieją głupie pytania

P: Czy możesz jeszcze raz powiedzieć, co to jest kolumna?

U: Kolumna to jedno pole w tabeli. A zatem w tabeli *People* możesz mieć kolumny *FirstName* oraz *LastName*, które będą oznaczały imię i nazwisko osoby. Z kolumną jest także nierozzerwalnie związany typ danych taki jak *String*, *Date* czy też *Bool*.

P: Do czego jest nam potrzebna kolumna *ContactID*?

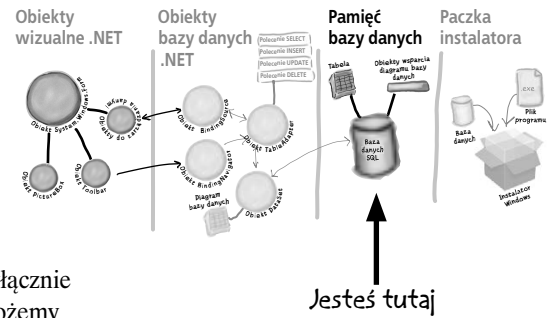
U: Umożliwia ona posiadanie unikalnego identyfikatora przez każdy rekord w większości tabel. Dlatego w przypadku przechowywania listy kontaktów do poszczególnych osób zdecydowaliśmy się utworzyć specjalną kolumnę i nazwać ją *ContactID*.

P: Co oznacza *Int* w polu *Data Type*?

U: Pole *Data Type* określa, jaki rodzaj informacji będzie przechowywany w danej kolumnie. *Int* jest skrótem od *integer*, co oznacza liczbę całkowitą. Na tej podstawie wnioskujemy, że kolumna *ContactID* będzie przechowywała liczby całkowite.

P: To strasznie dużo różnych informacji. Czy mam to wszystko rozumieć?

U: Nie, nie ma problemu, jeżeli wszystkiego w tej chwili nie rozumiesz. Twoim aktualnym zadaniem powinno być poznanie podstawowych sposobów korzystania z Visual Studio IDE w celu określenia postaci formularza i uruchamiania programu. (Jeśli nie możesz się doczekać, by dowiedzieć się czegoś więcej na temat baz danych, to zawsze możesz sięgnąć po książkę *SQL. Rusz głową!*).



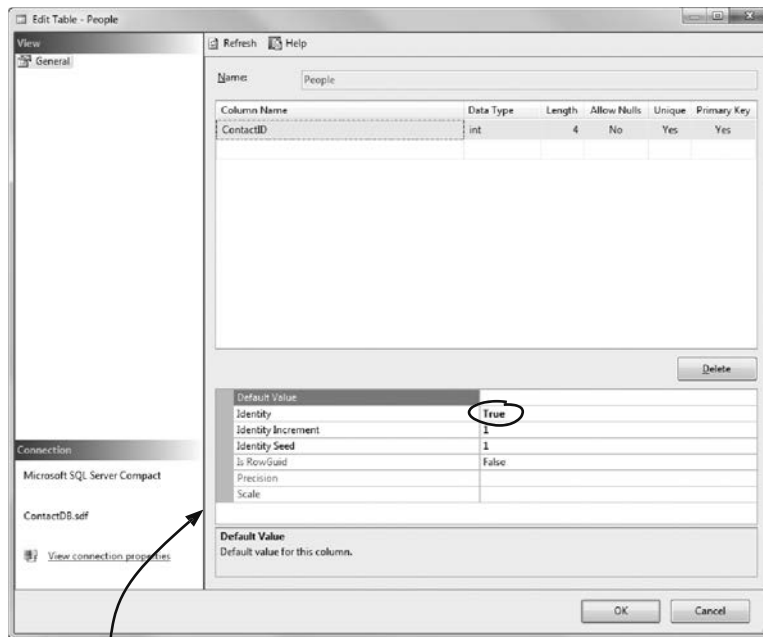
3 Powiedz bazie danych, aby automatycznie generowała identyfikatory.

Dopóki ContactID jest numerem wykorzystywanym wyłącznie przez bazę danych, a nie przez użytkownika, dopóty możemy obciążyć bazę obowiązkiem automatycznego generowania identyfikatorów. W ten sposób nie musimy się martwić o pisanie żadnego kodu potrzebnego do tego celu.

We właściwościach tabeli wyświetlonych poniżej przypisz wartość „True” właściwości *Identity*, dzięki czemu *ContactID* stanie się kolumną identyfikującą rekordy.

Nie zapomnij także wpisać nazwy tabeli, „People”, w polu *Name* u góry okna.

W tym oknie definiujesz swoją tabelę i dane, które będzie przechowywała.



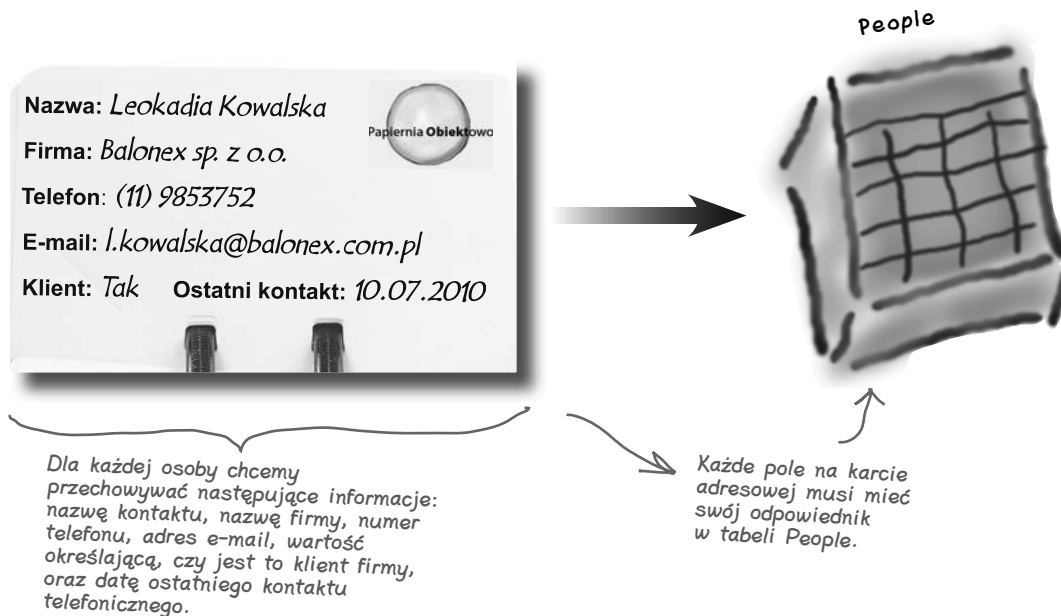
Klucz główny (ang. primary key) pomaga bazie danych szybko odnajdywać rekordy. Ponieważ jest on podstawowym narzędziem, którego program będzie używał do odnajdywania rekordów w bazie, zatem jego wartość zawsze musi być określona.

Dzięki temu pole ContactID aktualizuje się automatycznie za każdym razem, gdy jest dodawany nowy rekord.

Aby oznaczyć ContactID jako identyfikator rekordów w tabeli, będziesz musiał kliknąć w prawej kolumnie w wierszu Identity i wybrać opcję True z rozwijalnej listy.

Pola na karcie kontaktowej stają się kolumnami w tabeli People

Teraz, kiedy już utworzyłeś klucz główny tabeli, potrzebujesz zdefiniować pozostałe pola, które zamierzasz przechowywać w bazie danych. Każda rubryka w naszej pisanej karcie kontaktowej musi mieć swój odpowiednik w postaci kolumny tabeli People.



WYSIL SZARE KOMÓRKI

Jakie problemy mogą wynikać z przechowywania wielu rekordów dotyczących tej samej osoby?

* * ? * KTO CO ROBI?

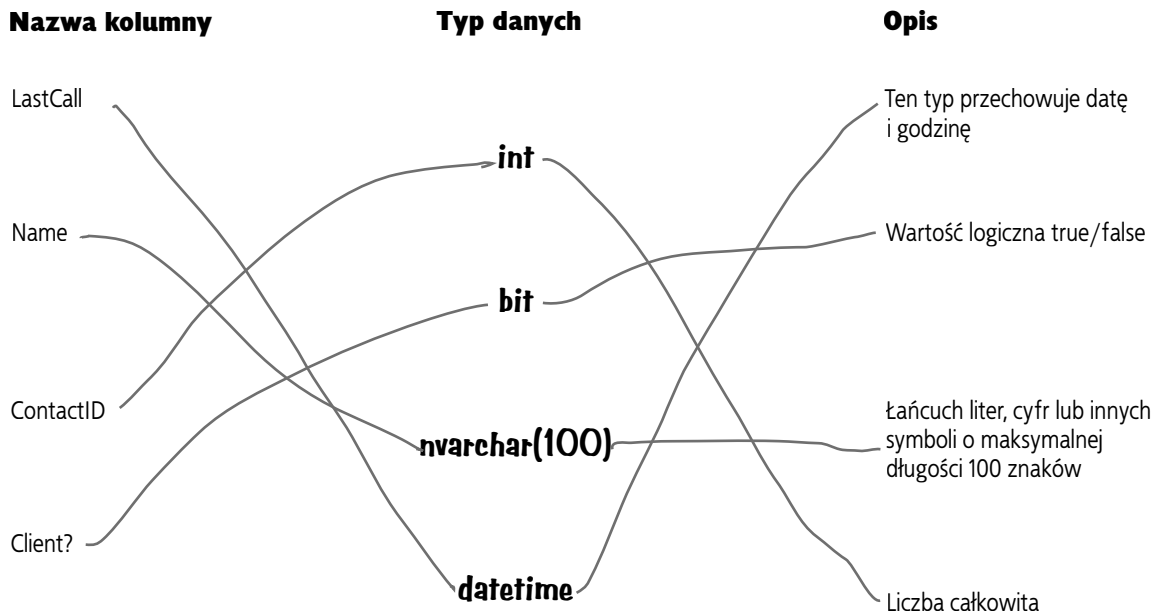
Teraz, kiedy stworzyłeś tabelę People oraz kolumnę z kluczem głównym, musisz dodać kolumny dla wszystkich pozostałych pól danych. Zobaczmy, czy potrafisz dopasować określone dane do odpowiedniej kolumny w tabeli oraz dobrać opis właściwy dla ich typu.

Nazwa kolumny	Typ danych	Opis
LastCall	int	Ten typ przechowuje datę i godzinę
Name	bit	Wartość logiczna true/false
ContactID	nvarchar(100)	Łańcuch liter, cyfr lub innych symboli o maksymalnej długości 100 znaków
Client?	datetime	Liczba całkowita

* * ? * KTO CO ROBI?

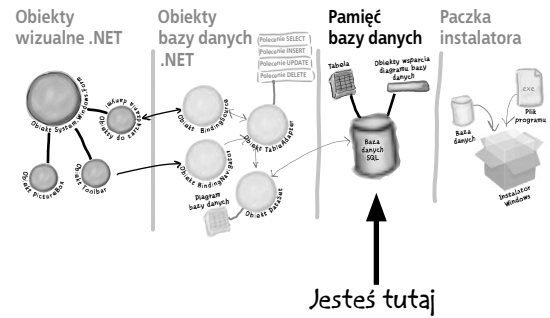
Rozwiązanie

Teraz, kiedy stworzyłeś tabelę People oraz kolumnę z kluczem głównym, musisz dodać kolumny dla wszystkich pozostałych pól danych. Zobaczmy, czy potrafisz dopasować określone dane do odpowiedniej kolumny w tabeli oraz dobrać opis właściwy dla ich typu.



Zakończ tworzenie tabeli

Wróć do miejsca, gdzie wpisywałeś nazwę kolumny ContactID, i dodaj sześć pozostałych kolumn z karty kontaktowej. Tak oto powinna wyglądać tabela bazy danych, kiedy skończysz:



Column Name	Data Type	Length	Allow Nulls	Unique	Primary Key
ContactID	int	4	No	Yes	Yes
Name	nvarchar	100	Yes	No	No
Company	nvarchar	100	Yes	No	No
Telephone	nvarchar	100	Yes	No	No
Email	nvarchar	100	Yes	No	No
Client	bit	1	Yes	No	No
LastCall	datetime	8	Yes	No	No

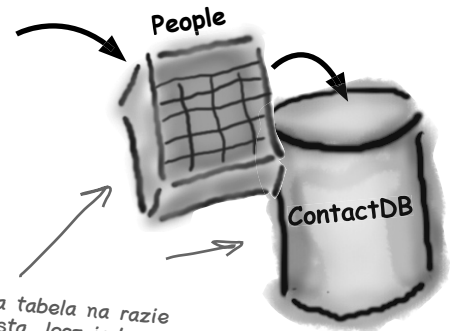
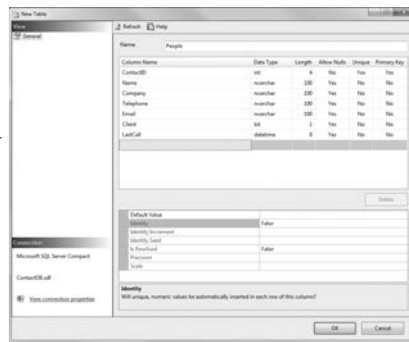
Pola bitowe przechowują wartości True lub False i mogą być reprezentowane przez kontrolkę CheckBox.

Jeżeli usuniesz zaznaczenie pola Allow Nulls, kolumna będzie musiała posiadać wartość.

Niektóre karty mogą zawierać niepełne informacje, więc niektóre kolumny pozostaną puste.

Kliknij przycisk OK, aby zapisać swoją nową tabelę. W ten sposób nowa pusta tabela zostanie dodana do bazy danych.

Po kliknięciu przycisku OK Visual Studio doda tabelę People do bazy danych.

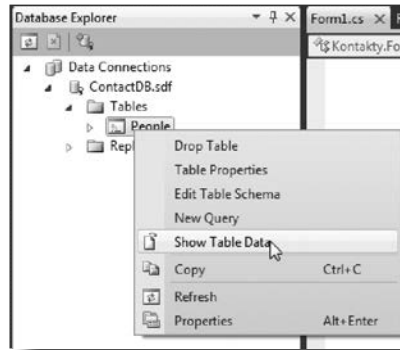


Ta nowa tabela na razie jest pusta, lecz jednocześnie jest już gotowa do wpisywania danych.

Wstaw dane z kart do bazy

Teraz jesteś już gotowy do wstawienia danych z kart do bazy. Mamy tutaj kilka kontaktów szefa — użyjemy ich do przygotowania wstępnej bazy danych zawierającej kilka rekordów.

- 1 W oknie *Database Explorer* (lub *Server Explorer*) rozwiń element *Tables*, kliknij tabelę *People* prawym przyciskiem myszy i wybierz *Show Table Data*.



Twoim zadaniem jest wpisanie danych z tych sześciu kart do tabeli *People*.

- 2 Gdy na głównym ekranie pojawi się tabela, wpisz do niej wszystkie poniższe dane. (Na początku będą tam same wartości null, ale nie przejmuj się. Zapełnij pierwszy rząd prawidłowymi danymi, ignorując ostrzeżenia pokazujące się obok nich). Nie musisz uzupełniać kolumny *ContactID*, bo zostanie to zrobione automatycznie.

Wpisz True lub False w kolumnie *Client*. To w tej postaci SQL przechowuje wartości typu „Tak” lub „Nie”.

Nazwa: Leszek Jawczyk
Firma: EkstraLans S.A.
Telefon: (12) 1234567
E-mail: leszekj@el_sa.pl
Klient: Tak **Ostatni kontakt:** 12.01.2010

Nazwa: Eliza Wizewska
Firma: EWAR
Telefon: (11) 1928375
E-mail: eliza_wizewska@ewar.org
Klient: Tak **Ostatni kontakt:** 30.08.2009

Nazwa: Krystyna Bryczewska
Firma: Dom Wydawniczy Bryczewska
Telefon: (48) 8721723
E-mail: k.bryczewska@bryczewska.pl
Klient: Tak **Ostatni kontakt:** 21.07.2010



Firma Papiernia Obiektowo znajduje się w Polsce, zatem dyrektor wpisuje daty w postaci 21.05.2010, gdzie 05 oznacza miesiąc maj. Jeśli system został skonfigurowany dla innego kraju, być może, że daty trzeba będzie wpisywać w innym formacie, na przykład 05/26/10.

- ③ Po wpisaniu sześciu rekordów wybierz po raz kolejny *Save All* z menu *File*. Powinno to zaowocować zapisaniem ich wszystkich w bazie.

Save All nakazuje IDE zapisać wszystko to, co należy do aplikacji. Tym różni się od zwykłego *Save*, zapisującego tylko plik, nad którym w danej chwili się pracuje.

Nie istnieją
głupie pytania

P: Co się dzieje z danymi po tym, jak je wpisałem? Gdzie one idą?

O: IDE automatycznie zapisuje wprowadzone wartości w tabeli *People* w bazie danych. Tabela — jej kolumny, typy i wszystkie dane jej dotyczące — przechowywana jest w pliku *ContactDB.sdf* SQL Server Compact. Plik ten jest integralną częścią projektu i IDE aktualizuje go podczas jego zmiany tak jak każdy inny plik z kodem.

P: Dobrze, wpisałem te sześć rekordów. Czy one będą częścią mojego programu na zawsze?

O: Tak, będą one integralną częścią tak samo jak kod, który napisałeś, i formularz, który zaprojektowałeś. Różnica polega na tym, że baza nie jest kompilowana do postaci programu wykonywalnego. Jej plik jest kopiowany i przechowywany razem z plikiem wykonywalnym. Gdy aplikacja potrzebuje dostępu do danych, wykonuje szereg operacji odczytu i zapisu na pliku *ContactDB.sdf* znajdującym się w jej katalogu wyjściowym.

Ten plik jest w rzeczywistości bazą danych SQL i Twój program może go używać za pomocą kodu wygenerowanego dla Ciebie przez IDE.



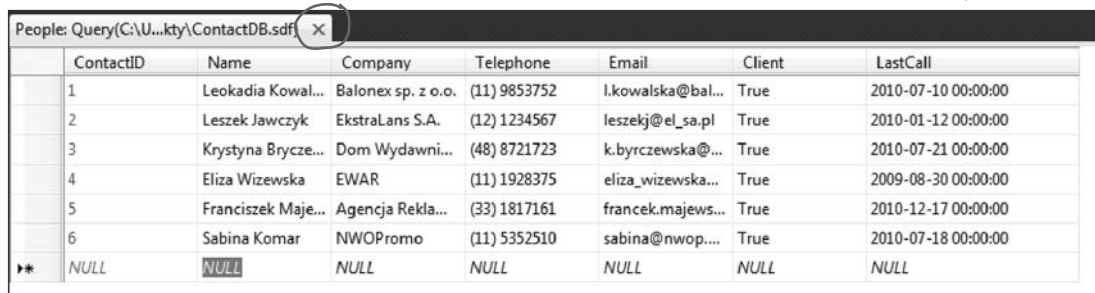
ContactDB.sdf

Połącz formularz z bazą danych, korzystając ze źródeł danych

Nareszcie jesteśmy gotowi do utworzenia obiektów .NET, które będą wykorzystywane przez formularz do integracji z bazą danych. Potrzebujemy **źródła danych**, które w rzeczywistości będzie kolekcją instrukcji SQL używanych do wymiany informacji pomiędzy formularzem a bazą *ContactDB*.

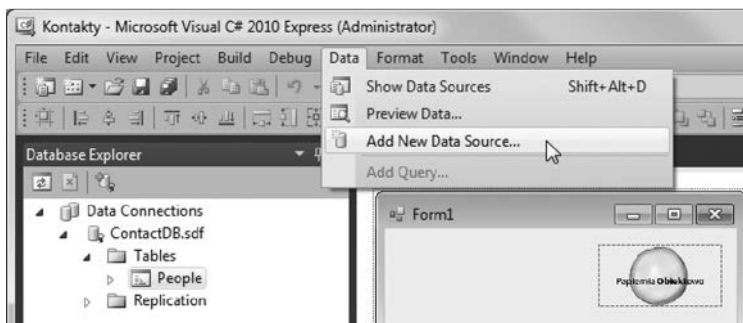
- 1 Przejdź z powrotem do formularza aplikacji.**
Zamknij tabelę *People* i diagram bazy danych *ContactDB*. Powinieneś w tej chwili widzieć zakładkę *Form1.cs [Design]*.

Kiedy już skończysz wprowadzać dane, zamknij okno danych, by wrócić do formularza.

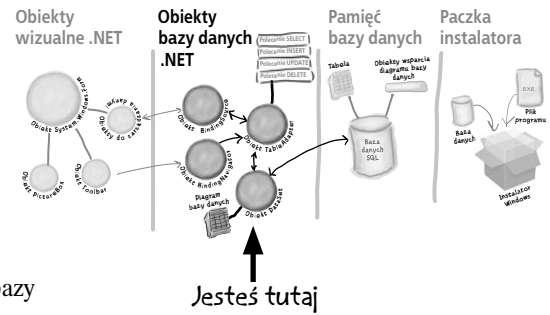


ContactID	Name	Company	Telephone	Email	Client	LastCall
1	Leokadia Kowal...	Balonex sp. z o.o.	(11) 9853752	l.kowalska@bal...	True	2010-07-10 00:00:00
2	Leszek Jawczyk	EkstraLans S.A.	(12) 1234567	leszekj@el_sa.pl	True	2010-01-12 00:00:00
3	Krystyna Brycze...	Dom Wydawni...	(48) 8721723	k.byrzczewska@...	True	2010-07-21 00:00:00
4	Eliza Wizewska	EWAR	(11) 1928375	eliza_wizewska...	True	2009-08-30 00:00:00
5	Franciszek Maje...	Agencja Rekla...	(33) 1817161	francek.majews...	True	2010-12-17 00:00:00
6	Sabina Komar	NWOPromo	(11) 5352510	sabina@nwop...	True	2010-07-18 00:00:00
▶*	NULL	NULL	NULL	NULL	NULL	NULL

- 2 Dodaj nowe źródło danych do aplikacji.**
W tej chwili powinno to być proste. Kliknij menu *Data*, a następnie wybierz z rozwiniętej listy *Add New Data Source...*



Źródło danych, które stworzysz, będzie obsługiwało całą komunikację pomiędzy formularzem i bazą danych.



3 Skonfiguruj źródło danych.

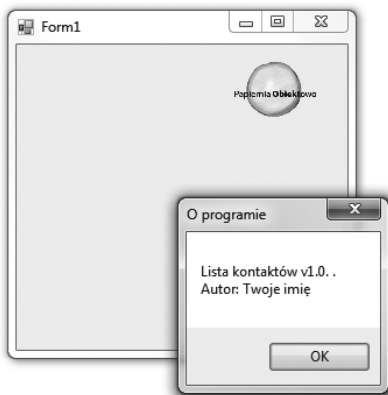
Teraz musisz dostosować źródło danych, aby używało bazy *ContactDB*. Poniżej opisano czynności, które w tym celu powinieneś wykonać:

- ◆ Krok 1.: Wybierz *Database* i kliknij przycisk *Next*.
- ◆ Krok 2.: Wybierz *Dataset* i kliknij przycisk *Next*.
- ◆ Krok 3.: Wybierz połączenie z bazą danych. Na rozwijalnej liście powinieneś już zobaczyć połączenie ze swoją bazą. Kliknij przycisk *Next*.
- ◆ Krok 4.: Na ekranie „Choose Your Objects” zaznacz pole wyboru przy elemencie *Tables*.
- ◆ Upewnij się, że w polu *DataSet Name* wpisane jest „*ContactDBDataSet*”, i kliknij *Finish*.

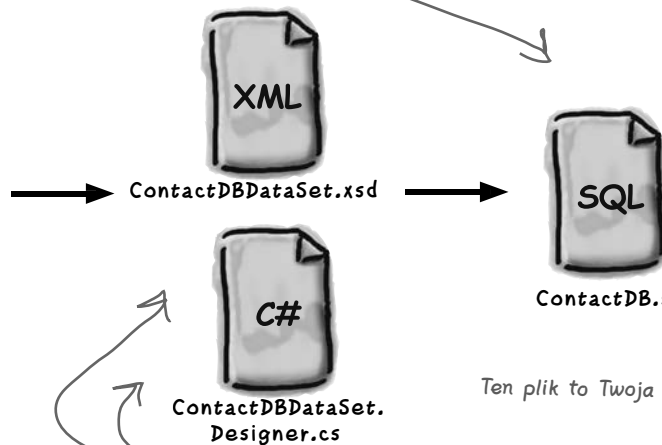
Przejdźcie przez poszczególne etapy pozwala na połączenie źródła danych z tabelą *People* w bazie danych *ContactDB*.

W wersjach Visual Studio innych niż Express zostaniesz poproszony o zapisanie połączenia w pliku konfiguracyjnym aplikacji. Kliknij przycisk „Yes”.

Teraz Twój formularz może używać źródła danych do interakcji z bazą danych *ContactDB*.



To jest Twój formularz.



Te pliki zostały wygenerowane przez źródło danych, które właśnie skonfigurowałeś.

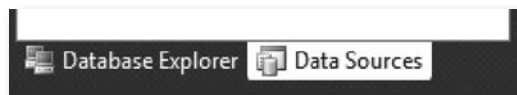
Ten plik to Twoja baza danych.

Dodaj kontrolki powiązane z bazą danych do formularza

Teraz możemy powrócić do naszego formularza i dodać kilka kontrolki. Nie będą to jednak zwykłe kontrolki, tylko takie, które będą *powiązane* z naszą bazą danych i kolumnami w tabeli *People*. Oznacza to, że każda zmiana danych w jednej z nich pociągnie za sobą automatyczną zmianę informacji w odpowiedniej kolumnie bazy danych.

A oto sposób na utworzenie kilku kontrolki powiązanych z bazą danych:

- 1 Wybierz źródło danych, którego chcesz używać.** Wybierz *Show Data Sources* z menu *Data*. Spowoduje to pojawienie się okna *Data Sources* z wyświetlonymi źródłami danych skonfigurowanymi dla tej aplikacji.



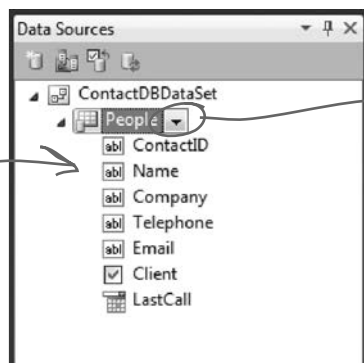
To okno pokazuje wszystkie Twoje źródła danych. W tej chwili skonfigurowane jest tylko jedno, ale oczywiście nie ma żadnych przeciwwskazań do posiadania większej ich liczby dla innych tabel lub baz danych.

Jeżeli nie widzisz tej zakładki, wybierz *Show Data Sources* z menu *Data*.

Możesz także poszukać zakładki *Data Sources* w dolnej części okna *Database Explorer*, a następnie ją kliknąć.

- 2 Wybierz tabelę *People*.**

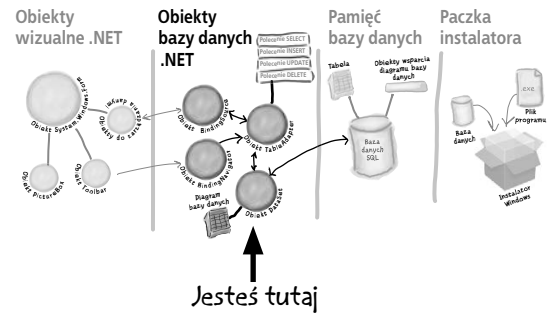
Pod elementem *ContactDBDataSet* powinieneś zobaczyć tabelę *People* i wszystkie jej kolumny. Kliknij strzałkę widoczną obok nazwy tabeli, aby wyświetlić jej zawartość — w ten sposób zobaczysz wszystkie kolumny dodane wcześniej. Kiedy w oknie *Data Sources* klikniesz tabelę *People* i przeciągniesz ją na formularz, IDE automatycznie doda do niego kontrolki, dzięki którym użytkownik będzie mógł przeglądać dane oraz je wpisywać. Domyślnie IDE dodaje kontrolkę *DataGridView*, przypominającą nieco okno arkusza kalkulacyjnego. Aby dodać do formularza osobne pola dla poszczególnych kolumn tabeli, kliknij strzałkę widoczną z jej prawej strony i z wyświetlonego menu wybierz opcję *Details*.



Tutaj powinny zostać wyświetlone wszystkie utworzone wcześniej kolumny tabeli.

Kliknij tę strzałkę i wybierz opcję *Details*, aby kazać IDE dodać do formularza osobne kontrolki dla poszczególnych kolumn, a nie jedną dużą kontrolkę przypominającą arkusz kalkulacyjny.

Tę rozwijalną listę zobaczysz wyłącznie w przypadku, gdy otworzone jest okno do projektowania formularzy. Pozwala ono na bezpośrednie **przeciągnięcie kontrolki danych** z okna źródła danych i umieszczenie ich na formularzu.



3 Utwórz kontrolki powiązane z tabelą People.

Przeciagnij i upuść tabelę *People* na formularz. Powinieneś zobaczyć wstawione kontrolki odpowiadające każdej kolumnie bazy danych. Nie przejmuj się w tej chwili ich wyglądem, tylko upewnij się, że wszystkie pojawiły się na formularzu.

Jeżeli coś przypadkowo kliknąłeś i z ekranu zniknął formularz, zawsze możesz do tego widoku powrócić, wybierając zakładkę Form1.cs [Design], lub otworzyć Form1.cs z okna Solution Explorer.

IDE tworzy ten pasek narzędzi do nawigacji po tabeli People.

Te elementy nie będą wyświetlane na formularzu, ale będą reprezentowały kod, który został utworzony przez IDE do zarządzania tabelą People oraz bazą danych ContactDB.

Obiekt peopleBindingNavigator łączy kontrolki paska narzędzi z tabelą bazy danych.

Ten obiekt łączy formularz z tabelą bazy danych.

Ten adapter pozwala kontrolkom wykorzystywać wyrażenia SQL, które IDE oraz źródło danych wcześniej wygenerowały.

Kiedy przeciągnąłeś tabelę People na formularz, dla każdej na formularz, dla każdej jej kolumny zostały utworzone odpowiednie kontrolki.

Dobre programy są intuicyjne w użyciu

Co prawda formularz teraz działa, ale nie wygląda dobrze. Twoja aplikacja musi być więcej niż tylko funkcjonalna. Powinna być łatwa w użyciu. Dzieli Cię zaledwie kilka etapów od osiągnięcia wyglądu formularza zbliżonego do wyglądu papierowych kart, których używaliśmy na początku niniejszego rozdziału.

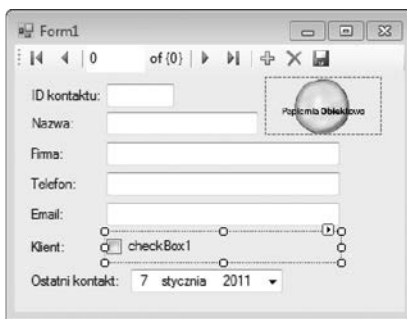
Nasz formularz byłby intuicyjny, gdyby wyglądał jak karta kontaktowa.



① Wyrównaj etykiety i pola tekstowe.

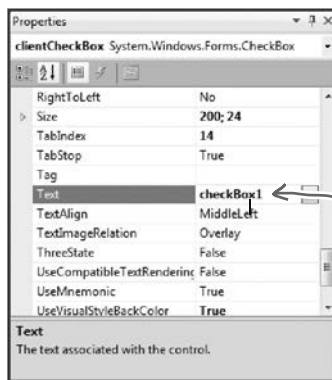
Wyrównaj etykiety i pola tekstowe względem lewej krawędzi formularza. Będzie on wyglądał podobnie do innych aplikacji, a użytkownicy będą czuli się znacznie bardziej komfortowo podczas korzystania z programu. Przetłumacz angielskie treści etykiet powstałych automatycznie na podstawie nazw kolumn z tabeli *People*.

Podczas przeciągania kontrolki na formularzu będą pojawiały się niebieskie linie. Są one po to, aby ułatwić wyrównywanie poszczególnych elementów interfejsu.

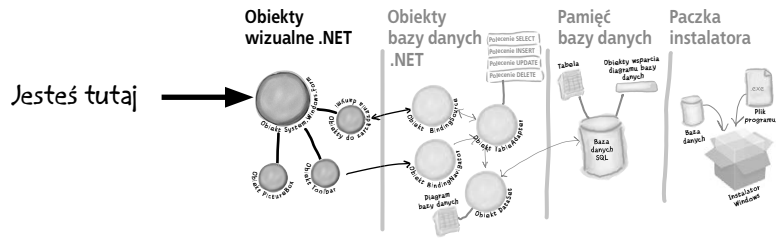


② Zmień właściwość Text kontrolki CheckBox.

Podczas wstawiania kontrolki do formularza obok kontrolki *CheckBox* pojawia się etykieta, która musi zostać usunięta. Po prawej stronie, pod oknem *Solution Explorer*, powinieneś dostrzec okno *Properties*. Przewiń jego zawartość w dół aż do napotkania właściwości *Text*. Usuń fragment „checkbox1”.



Usuń ten napis, aby pozbyć się tekstu z kontrolki *CheckBox*.



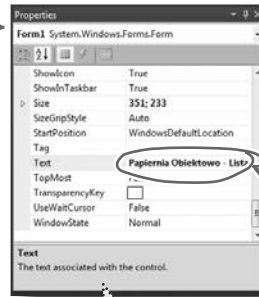
③ Nadaj aplikacji profesjonalny wygląd.

Możesz także zmienić nazwę formularza. Dokonasz tego poprzez kliknięcie w jego dowolnym pustym miejscu i odnalezienie właściwości *Text* w oknie *Properties*. Dla naszych potrzeb zmień nazwę na „Papiernia Obiektowo — Lista kontaktów”.

Możesz także, w tym samym oknie, wyłączyć przyciski do minimalizacji i maksymalizacji. W tym celu poszukaj właściwości *MaximizeBox* oraz *MinimizeBox* i ustaw ich wartości na *False*.

Powodem, dla którego chcesz usunąć przycisk maksymalizacji, jest to, że nie zmienia ona pozycji kontrolki na formularzu. Pusta przestrzeń sprawia, że wygląda on dość dziwnie.

Okno Properties powinno się znajdować zaraz pod oknem Solution Explorer, w dolnej części prawego panelu IDE.



Właściwość Text określa tekst wyświetlany na pasku tytułowym formularza.

Jeżeli nie masz okna Properties, to możesz je uaktywnić, wybierając odpowiednią pozycję z menu View.

Dobra aplikacja to nie tylko taka, która działa, ale taka, która jest także łatwa w użyciu. Dobrym zwyczajem programisty jest upewnienie się, że aplikacja zachowuje się dokładnie tak, jak spodziewałby się tego przeciętny użytkownik.

Jazda próbna

Pozostała nam do zrobienia tylko jedna rzecz: uruchomić program i sprawdzić, czy wszystko działa tak, jak powinno. Zrób to tak samo jak do tej pory — naciśnij klawisz F5 na klawiaturze albo kliknij w przycisk z zieloną strzałką (▶) na pasku narzędzi (lub wybierz *Run* z menu *Debug*).

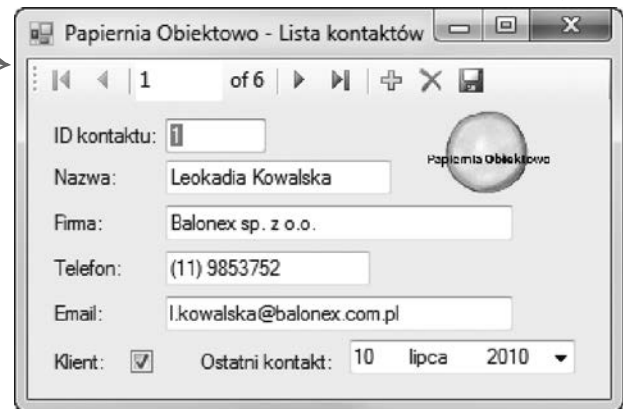
Możesz uruchomić swój program w każdej chwili, nawet jeśli nie jest jeszcze skończony. Jeżeli w kodzie są błędy, to IDE Ci o tym przypomni i wstrzyma wykonywanie programu.

Naciśnij ten X w rogu, aby zamknąć program. W ten sposób możesz przejść do następnego etapu.

Budowanie programu nadpisuje dane w bazie.

Poświęcimy na to więcej czasu w następnym rozdziale.

Te kontrolki pozwalają przeglądać kolejne rekordy zapisane w bazie danych.



IDE najpierw buduje, potem wykonuje.

Kiedy uruchamiasz swoją aplikację, IDE właściwie robi dwie rzeczy. Najpierw **buduje** program, a następnie go **wykonuje**. Proces ten składa się z kilku niezależnych części. IDE **kompiluje** kod i zamienia go na postać wykonywalną. W następnej kolejności umieszcza skompilowany kod razem z zasobami i innymi plikami w podkatalogu katalogu *bin*.

W tym przypadku plik wykonywalny oraz baza danych SQL znajdują się w katalogu *bin/debug*. Plik bazodanowy jest kopiowany za każdym razem, więc zmiany wprowadzone w trakcie pracy z IDE zostaną utracone. Jeżeli plik wykonywalny będzie uruchamiany bezpośrednio z poziomu systemu Windows, dane zostaną zachowane — chyba że, po raz kolejny, projekt zostanie zbudowany przy użyciu IDE. W takim przypadku IDE nadpisze bazę danych SQL nową kopią zawierającą dane skonfigurowane przy pomocy okna *Database Explorer*.



Za każdym razem, gdy budujesz program, IDE umieszcza w katalogu bin nową

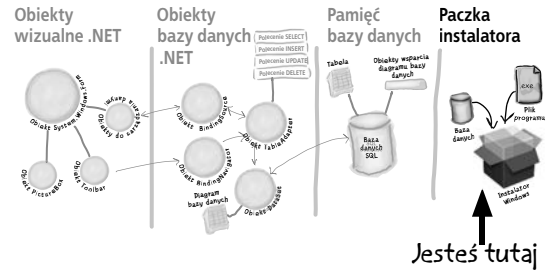
kopię bazy danych. Oznacza to utratę wszelkich danych zapisanych w niej podczas wcześniejszego uruchomienia programu.

Jeśli kod programu się zmienił, to próba debugowania go z poziomu IDE powoduje jego ponowne zbudowanie — to z kolei oznacza, że czasami uruchomienie programu z poziomu IDE spowoduje nadpisanie wcześniejszej zawartości bazy. Jeśli jednak uruchomisz swój program bezpośrednio z katalogu bin/debug, bądź jeśli użyjesz instalatora, by go zainstalować na swoim komputerze, to problem utraty danych nie wystąpi.

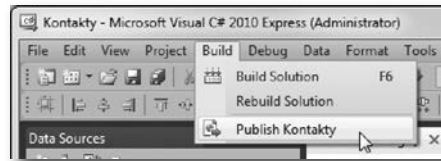
Jak zamienić TWOJĄ aplikację w aplikację WSZYSTKICH

W tym momencie posiadasz wspaniały program. Problem polega na tym, że działa on tylko na Twoim komputerze. Oznacza to, że nikt inny go nie będzie używał, nikt Ci za niego nie zapłaci, nikt nie zobaczy, jak wspaniałym programistą jesteś, i nikt Cię nie zatrudni... a Twój szef i klienci nie dostaną raportów generowanych z bazy danych.

C# znacznie ułatwia przeprowadzenie procesu **wdrożenia**. Wdrożenie aplikacji to zainstalowanie jej na innych komputerach. Z pomocą Visual C# IDE przygotowanie sprowadza się do dwóch etapów.



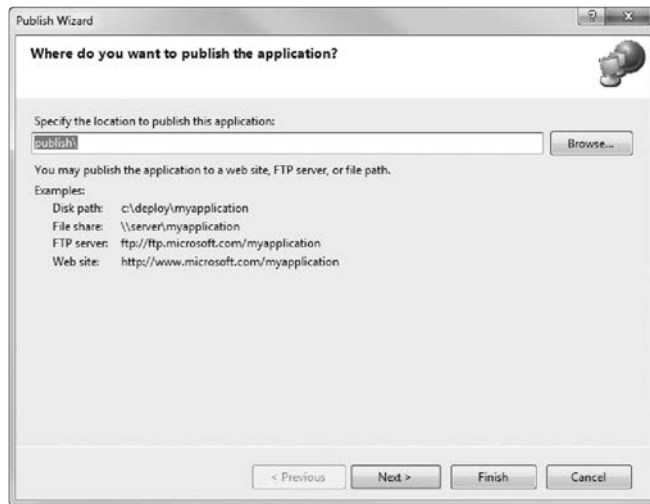
- 1 Wybierz *Publish Kontakty* z menu *Project*.



Budowanie projektu powoduje kopiowanie plików na komputer lokalny. Opcja *Publish* tworzy instalacyjny plik wykonywalny wraz z plikiem konfiguracyjnym, aby inni użytkownicy także mogli zainstalować Twój program.

- 2 Zaakceptuj wszystkie domyślne wartości w oknie *Publish Wizard* poprzez naciśnięcie przycisku *Finish*. Zobaczysz, jak Twoja aplikacja będzie pakowana, a następnie ujrzysz folder, w którym znajdzie się plik *setup.exe*.

Jeśli używasz *Visual Studio* w wersji *Express*, to opcję *Publish* znajdziesz w menu *Project*; w innych wersjach *VS* została ona umieszczona w menu *Build*.



Przekaż aplikację innym użytkownikom

Z chwilą utworzenia paczki instalacyjnej utworzony został także folder *publish/*. Katalog zawiera kilka składników, a wszystkie związane są z procesem instalacji. Najważniejszym dla użytkowników jest *setup*, czyli program, który pozwoli im zainstalować Twoją aplikację na ich własnych komputerach.

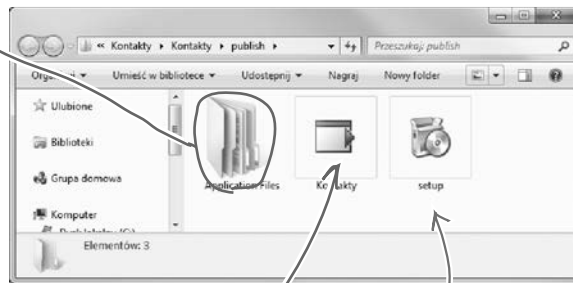


**Być może
będziesz
musiał
uruchamiać**

**program instalacyjny
z prawami administratora.**

Jeśli na komputerze nie jest jeszcze zainstalowany SQL Server Compact, to wygenerowany program instalacyjny spróbuje go pobrać i zainstalować. Na niektórych komputerach operacji tych nie da się wykonać, jeśli program nie zostanie uruchomiony przez użytkownika z prawami administratora, dlatego też kliknij jego ikonę prawym przyciskiem myszy i z menu podręcznego wybierz opcję „Uruchom jako administrator”. Jeśli nie masz do niej dostępu, to nie musisz się tym przejmować — i bez niej możesz kontynuować lekturę tej książki.

To jest miejsce, gdzie są przechowywane wszystkie pomocnicze pliki instalatora.



Ten plik udziela instalatorowi wszelkich informacji na temat tego, które pliki powinny zostać dołączone do programu podczas instalacji.

W ten sposób użytkownicy Twojego programu będą go instalować na swoich komputerach!

Moja sekretarka właśnie mi powiedziała, że skończyła pisać nową bazę kontaktową i że wszystko już działa. Pakuj swoje walizki — mamy miejsce w odrzutowcu do Aspen dla takiego pomysłowego pracownika jak Ty!

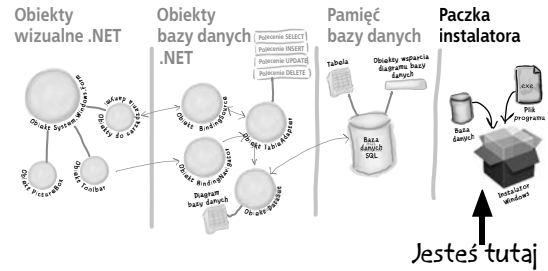


Wydaje się, że szef jest zadowolony. Dobra robota! Jednak zanim polecisz szusować na stokach, pozostaje jeszcze jedna rzecz do zrobienia...

Jeszcze nie skończyłeś: przetestuj instalację

Zanim zaczniesz strzelać korkami od szampana, musisz przetestować wdrożenie i instalację. Nie dasz chyba nikomu swojego programu, dopóki sam go nie wypróbujesz?

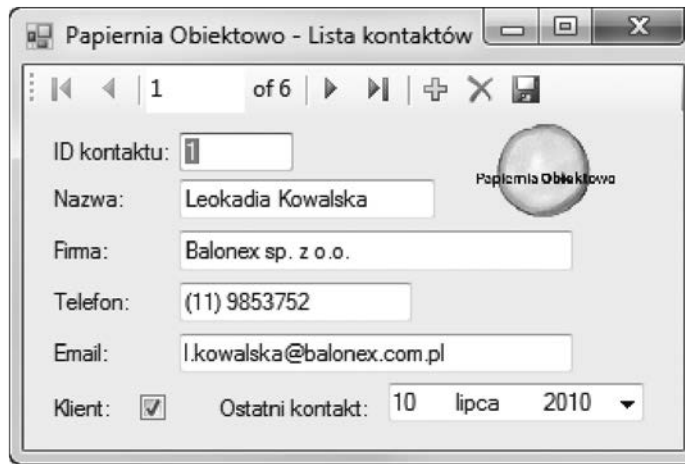
Zamknij Visual Studio IDE. Uruchom program instalacyjny i wybierz katalog na swoim komputerze, w którym chcesz umieścić aplikację. Następnie uruchom program z podanej lokalizacji i upewnij się, że działa dokładnie tak, jak sobie wyobrażałeś. Możesz także dodać lub zmodyfikować kilka rekordów. Zostaną one oczywiście zapisane w bazie danych.



Teraz możesz dodawać, zmieniać i usuwać rekordy. Wszystko to zostanie zapisane w bazie danych.

Możesz używać strzałek i pola tekstowego do poruszania się pomiędzy rekordami.

Nie bój się... Wprowadź kilka zmian. Aplikacja jest zainstalowana, więc tym razem dane nie zginą.



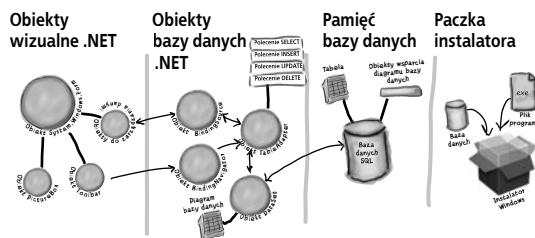
Znajdziesz tu wszystkie wpisane wcześniej kontakty. Są one zapisane w pliku bazy danych — ContactDB.sdf — który został zainstalowany wraz z Twoim programem.

TESTUJ WSZYSTKO!

Przetestuj swój program,
przetestuj swoją instalację,
sprawdź dane w swojej aplikacji.

Stworzyłeś pełnowartościową aplikację bazodanową

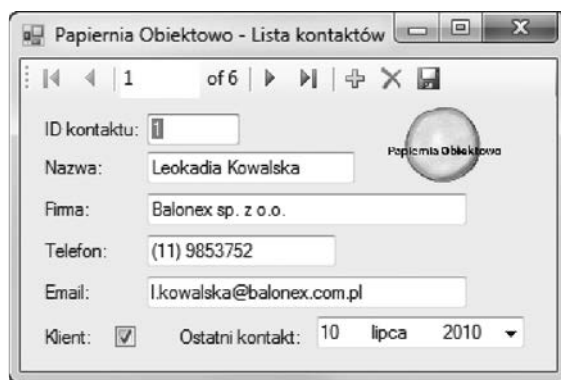
Dzięki Visual Studio IDE stworzenie aplikacji Windows, zaprojektowanie i utworzenie bazy danych oraz połączenie ich w jedną zgrabną całość jest dość proste. Możesz nawet wygenerować program instalacyjny za pomocą zaledwie kilku kliknięć.



Od tego



do tego



w mgnieniu oka.

Siła Visual C# polega na tym, że możesz szybko utworzyć coś działającego, a następnie skupić się na tym, co rzeczywiście Twój program powinien robić... a nie na oprogramowywaniu liczących okien, przycisków i zestawu instrukcji SQL.

Skorowidz

--, 100
!=, 106, 774
&, 762
&&, 106
*=", 172
.NET, 82, 767
.NET Framework, 788
.NET Framework 4.0, 36
///, 758
?., 761
??, 761
@, 449
[DllImport], 676
[NonSerialized], 606
[Serializable], 469, 470, 485
|, 762
||, 106, 761
~, 762
+, 100, 165
++, 100, 761
<<, 763
=, 105, 761
-=", 172
==, 105, 106, 772, 774
=>, 781
>>, 763

A

abstract, 328, 329
abstrakcja, 336
Add Existing Item, 294, 728
Add New Data Source, 68
Add New Item, 58
Add Project, 766
Add Reference, 275, 765
Add Resource, 622
Add Watch, 102
Add(), 364, 368, 392, 618, 716
Add/New Project, 766
akcesory, 235, 241
 get, 235, 241
 set, 235, 237, 241
aktualizowanie kod, 297
aktywacja punktu
 przerwania, 102
aktywność mózgu, 33
AND, 762
animacja, 622, 741
 podwójne buforowanie, 656
animowana kontrolka, 621

animowany symulator ula, 565
animacja, 614
architektura
 symulatora, 566
Bee, 572
Field, 618
formularz, 592
Go(), 581, 590
grupy pszczoł, 602
Hive, 577, 579, 580, 586,
 589, 618
interfejs użytkownika, 614
kwiaty, 571
Render(), 617
rendering, 616
statystyki, 593
system turowy, 583
świat, 582
tury, 583
World, 582, 584, 586, 589,
 593, 617
zegary, 596
anonimowe typy, 780
aplikacje, 43
 aplikacje bazodanowe, 78
 aplikacje konsolowe,
 276, 760
 aplikacje odporne, 696
 aplikacje WPF, 786
 punkt wejścia, 90
 testowanie, 77
 uruchamianie, 56
aplikacje Windows, 80
 formularze, 80
 projekt, 80
Append(), 763
AppendAllText(), 448
AppendFormat(), 763
AppendLine(), 763
Application.DoEvents(),
 114, 768
architektura aplikacji, 206
args, 482
ArgumentNullException, 516
argumenty, 167
 argumenty nazwane, 694
 argumenty opcjonalne, 694
 argumenty wywołania
 programu, 482
 argumenty zdarzenia, 534
Array.Resize(), 362

Array.Reverse(), 474
as, 313, 316, 318, 690
ASCII, 483
assembly, 764
atrybuty, 469
 [DllImport], 676
 [NonSerialized], 606
 [Serializable], 469, 470, 485
automatyczne konwersje typów,
 165, 166
AutoSize, 456
Average(), 718

B

BackColor, 113, 415, 624
BackgroundImage, 624
BackgroundWorker, 768, 770
badanie wyjątków, 505
bajt, 474
base, 282
BaseType, 771
baza danych, 43, 57, 58, 605, 727
 klucz główny, 61
 połączenie, 68
 SQL, 58, 59, 733
 tabele, 60
 wstawianie danych, 66
 źródło danych, 68
biblioteki, 764
BinaryFormatter, 468, 471,
 502, 681
BinaryReader, 476
 ReadBytes(), 476
 ReadChar(), 476
 ReadInt32(), 476
 ReadSingle(), 476
 ReadString(), 476
BinaryWriter, 475
Birthday.Value, 696
bit, 63
Bitmap, 640, 641, 651, 654, 659
bitmapy, 640
blok catch, 503, 505, 510, 522
blok finally, 508, 509
blok kodu, 94, 251
blok try, 503
blok using, 521
błędy, 50, 94, 491, 501
 błędy kompilatora, 85
 błędy systemowe, 454

bool, 160
BorderStyle, 456
break, 461, 760
Brush, 644
Brushes, 644
budowa programu, 88
budowanie projektu, 74, 75
Button, 619
by, 730
byte, 160, 163

C

C#, 36, 41, 42, 43, 788
CanOverflow, 607
case, 461
catch, 503, 505, 522
char, 161, 163, 473
CheckBox, 72, 254
 Text, 72
Checked, 113
class, 89, 197
Class Library, 764
Clear(), 716
Click, 546, 644, 649
Clone, 678, 680, 681
Close(), 437, 441, 449, 484
CLR, 83, 188, 676, 767
Color.FromArgb(), 114
Color.Transparent, 415, 637
ComboBox, 291, 341
Common Intermediate
 Language, 767
Common Language
 Runtime, 83
CompareTo(), 375
Console, 437
 ReadKey(), 375, 379,
 380, 763
 Write(), 473
 WriteLine(), 236, 379
Console Application, 276,
 367, 549
const, 214
ContactDB, 733
ContactDB SQL, 732
ContactID, 60, 61
Contains(), 364, 368, 644,
 645, 716
continue, 760
Continue, 497

- Control, 624
 - Controls, 613
 - Add(), 620
 - Remove(), 619
 - Convert
 - ToDateTime(), 457
 - ToInt32(), 762
 - ToString(), 762
 - CopyTo(), 716
 - Count, 364, 368, 392
 - Create(), 448, 471, 477
 - CreateDirectory(), 448
 - CreateGraphics(), 641, 642, 643, 654
 - CryptoStream, 442
 - csproj, 82
 - czarna skrzynka, 231
 - czas, 457, 596
 - czas wykonywania, 504
 - zczcionki, 643, 645
- ## D
- dane, 57, 98, 111, 355, 605, 707, 709
 - dane binarne, 479
 - dane XML, 784
 - Data Source Configuration Wizard, 58, 728
 - Data Type, 60
 - Database Explorer, 58, 60, 74
 - DataContractSerializer, 782, 783
 - DataGridView, 70
 - datetime, 63
 - DateTime, 596, 695
 - Now, 525
 - Parse(), 525
 - TryParse(), 696
 - DateTimePicker, 457, 525
 - Debug, 56, 497
 - Debug/Continue, 102
 - Debug/Step Over, 102
 - debugger, 101, 367, 493, 497, 498
 - debugowanie, 56, 83, 102
 - decimal, 161, 217
 - definiowanie
 - interfejsy, 303
 - zdarzenia, 534, 544
 - deklaracje, 90
 - delegaty, 549
 - interfejsy, 304
 - klasy abstrakcyjne, 328
 - stałe, 214
 - zdarzenia, 537, 544
 - zmienne, 98
 - zmienne referencyjne, 183
 - delegate, 549, 550
 - delegaty, 548, 549, 550, 557, 558
 - działanie, 551
 - funkcje zwrotne, 556
 - DELETE, 46
 - Delete(), 448
 - Descendants(), 785
 - descending, 718
 - deserializacja, 464, 465, 467, 485, 607
 - Deserialize(), 468, 471, 502, 507
 - destruktor, 676
 - diagramy klas, 142, 144
 - DialogResult, 445, 447
 - Dictionary, 391
 - Directory, 448, 450
 - CreateDirectory(), 452
 - Exists(), 452
 - GetFiles(), 495
 - SetCreationTime(), 452
 - Dispose(), 453, 454, 521, 625, 627, 678, 680, 681, 682
 - serializacja, 681
 - DivideByZeroException, 491, 496
 - DLL, 767
 - długość tablicy, 184
 - dobre aplikacje, 73
 - Dock, 193, 451
 - dodawanie kontrolki do formularza, 612
 - dodawanie kontrolki do okna Toolbox, 626
 - dodawanie tekstu do pliku, 448
 - dodawanie zachowań, 297
 - dodawanie źródła danych, 68
 - DoEvents(), 768
 - dokumenty XML, 604, 784
 - dopełnienie wartości, 762
 - dostęp do klasy bazowej, 282
 - double, 160
 - DoubleBuffered, 657, 659, 745
 - DoWork, 768
 - DragEventArgs, 537
 - DragEventHandler, 537
 - DrawEllipse(), 643
 - DrawImage(), 640, 641, 644, 652
 - DrawImageUnscaled(), 656
 - DrawLine(), 643
 - DrawLines(), 642, 645
 - DrawOneFrame(), 656
 - DrawRectangle(), 643
 - DrawString(), 643
 - DropDownList, 291, 341
 - DropDownStyle, 341
 - drukowanie, 662
 - dokumenty wielostronicowe, 663
 - Graphics, 662
 - HasMorePages, 663
 - obsługa zdarzeń, 662
 - podgląd wydruku, 663, 664
 - Print(), 662
 - PrintDialog, 663
 - PrintDocument, 662, 663
 - PrintPage, 664
 - PrintPreviewDialog, 663
 - PrintTableRow(), 664
 - dwukropek, 266
 - dziedziczenie, 247, 257, 258, 301, 336, 622
 - dodawanie zachowań, 297
 - dostęp do klasy bazowej, 282
 - dziedziczenie wielokrotne, 334
 - dziedziczenie zachowania, 280
 - hierarchia klas, 259, 264
 - interfejsy, 302, 311
 - klasy bazowe, 258, 261
 - klasy potomne, 261
 - konstruktor, 283
 - metody wirtualne, 270
 - podklasy, 258
 - przesłanie, 262, 270, 278, 280
 - tablica metod wirtualnych, 275
 - ukrywanie metod, 278
 - wywołanie metod, 265
 - wywołanie ukrytych metod, 279
 - dzielenie tekstu, 463
- ## E
- edycja formularza, 55
 - edytor XML, 787
 - efekt cienia, 643
 - efekty uboczne, 177
 - elementy tablicy, 183
 - else, 105, 197, 460
 - Encoding.BigEndian Unicode, 484
 - Encoding.UTF8, 483
 - EndOfStream, 441
 - enum, 357, 371
 - Enum.Parse(), 693
 - Enum.TryParse(), 693
 - Enumerator, 383
 - enumerator, 776
 - environment.NewLine, 401
 - equals, 725, 730
 - Equals(), 772, 773
 - Error List, 49, 50, 85, 94
 - etykiety, 72, 150, 620, 761
 - event, 534, 558
 - EventArgs, 534, 536
 - EventHandler, 534, 537, 544, 557
 - Events, 768
 - Exception, 492, 496, 510, 513, 519
 - EXE, 83, 764, 767
 - Exists(), 448
- ## F
- false, 100, 160
 - File, 48, 448, 450, 484
 - Copy(), 452
 - Create(), 471, 477
 - Delete(), 452
 - Encrypt(), 452
 - Exists(), 510
 - GetCreationTime(), 452
 - OpenWrite(), 477
 - ReadAllBytes(), 473, 474
 - ReadAllText(), 451
 - SetLastWriteTime(), 452
 - WriteAllBytes(), 473, 474, 482
 - WriteAllText(), 451, 452
 - FileInfo, 448, 484
 - FileNotFoundException, 510
 - FileStream, 435, 436, 449, 484
 - FillEllipse(), 643
 - FillRectangle(), 643, 644
 - Filter, 451
 - finalizatory, 676, 680, 682, 683
 - wyjątki, 683
 - wywołanie, 677
 - finally, 508, 509, 519
 - Fixed3D, 193, 456
 - float, 160, 161, 165
 - FlowLayoutPanel, 451
 - Font, 643, 645
 - FontStyle, 643
 - for, 103, 107, 197, 490
 - foreach, 367, 368, 382, 383, 585, 632, 718, 749
 - Form, 188, 613, 634

- Form Designer, 86, 87, 193
 - format szesnastkowy, 479
 - Format(), 480
 - FormatException, 496
 - FormBorderStyle, 193
 - formularze, 54, 55, 80, 150, 187, 226, 341, 451, 634, 635
 - dodawanie kontrolki, 612
 - konstruktory, 240
 - kontrolki, 613
 - obsługa, 254
 - podwójne buforowanie, 657
 - przerysowanie, 347, 650
 - przyciski maksymalizacji i minimalizacji, 193
 - tło, 639
 - wstawianie kontrolek, 86
 - zdarzenia, 545
 - from, 717, 718
 - FromImage(), 640, 654
 - FullName, 771
 - funkcje obsługi zdarzeń, 215, 226, 546
 - funkcje zwrotne, 529, 554, 557, 558
 - delegaty, 556
- ## G
- GAC, 765
 - GC.Collect(), 677, 679, 680, 683
 - GDI+, 642, 643
 - generowanie
 - widok szesnastkowy, 482
 - zdarzenia, 531
 - get, 235, 241, 500
 - GetEnumerator(), 383
 - GetFiles(), 448, 495
 - GetHashCode(), 772, 773
 - GetLastAccessTime(), 448
 - GetLastWriteTime(), 448
 - GetType(), 771
 - GetValueOrDefault(), 695
 - Global Assembly Cache, 765
 - Go To Definition, 453, 595, 731
 - goto, 761
 - gra przygodowa, 412
 - graficzny interfejs użytkownika, 52, 208
 - grafika, 611, 640, 745
 - Bitmap, 641
 - efekt cienia, 643
 - GDI+, 643
 - rysowanie, 616, 642
 - System.Drawing, 642
 - tekst, 643
 - zmiana rozmiaru bitmap, 640
 - Graphical User Interface, 131
 - Graphics, 640, 644, 650, 654, 745
 - FromImage(), 640
 - GripStyle, 607
 - group, 730
 - GroupBox, 209, 409
 - GUI, 131, 770
 - GZipStream, 435
- ## H
- HasMorePages, 663
 - Height, 624
 - heisenbug, 500
 - hermetyzacja, 211, 221, 223, 224, 229, 231, 244, 336, 485, 698
 - akcesory, 235
 - inicjalizacja pól prywatnych, 239
 - stosowanie, 233
 - hex, 479
 - hex dump, 479
 - hexdumper, 482
 - hierarchia klas, 259, 264, 301
- ## I
- ICollection<T>, 716, 775
 - IComparable<T>, 375
 - IComparer, 376, 378
 - IComparer<T>, 374, 378
 - IDE, 36, 41, 42, 43, 48, 51, 564
 - IDisposable, 453, 520, 625, 643, 678, 680
 - IEnumerable, 384, 405, 408
 - IEnumerable<T>, 383, 400, 710, 711, 716
 - IEnumerator, 776
 - IEnumerator<T>, 776
 - IEquatable, 772
 - IEquatable<T>, 772
 - if, 105, 167, 197, 460
 - else, 105
 - IL, 767
 - Image, 54
 - implementacja interfejsu, 305, 312
 - importowanie zasobów, 54
 - indeks tablicy, 183
 - indeksatory, 776
 - IndexOf(), 364, 368
 - IndexOutOfRangeException, 492, 496
 - informacje o pliku, 448
 - informacje o wyjątku, 513
 - inicjalizacja obiektów, 153
 - pola prywatne, 239
 - inicjalizatory, 238
 - inicjalizatory kolekcji, 372
 - inicjalizatory obiektów, 153
 - inicjowanie zdarzeń, 532
 - InitialDirectory, 451
 - InitializeComponent(), 188, 214, 240, 539
 - inkrementacja, 761
 - INSERT, 46
 - Insert Breakpoint, 101
 - Insert(), 368
 - instalacja aplikacji, 76
 - instalacja Visual Studio 2010, 36
 - instalator Windows, 47
 - instancja klasy, 130
 - instrukcje, 89, 104
 - SQL, 59
 - int, 63, 160, 161, 163, 173
 - IntelliSense, 85, 153, 767
 - interakcja z programami, 612
 - interfejs, 303
 - interfejs użytkownika, 52, 208, 253
 - interfejsy, 302, 306, 307, 318, 324, 332, 701
 - definiowanie, 303
 - diagramy klas, 311
 - dziedziczenie, 311
 - ICollection<T>, 716, 775
 - IComparable<T>, 375
 - IComparer, 376
 - IDisposable, 453, 520, 678
 - IEnumerable, 408
 - IEnumerable<T>, 383, 711, 716
 - IEquatable<T>, 772
 - implementacja, 305, 312, 318
 - nazwy, 303
 - referencje, 308, 309
 - rzutowanie w dół, 317
 - rzutowanie w górę, 317
 - stosowanie, 304
 - internal, 321, 766
 - Invaders, 735, 736
 - animacja, 741
 - architektura gry, 738
 - Draw(), 747, 751
 - FireShot(), 747
 - formularz gry, 740
 - Game, 746, 747
 - GameOver, 743
 - Go(), 744, 747
 - grafika, 745
 - gwiazdy, 754
 - Invader, 750
 - InvaderImage(), 751
 - kolizje, 749
 - komunikaty klawiatury, 742
 - LINQ, 749
 - misja, 737
 - Move(), 751
 - MovePlayer(), 747
 - NextWave(), 748
 - obsługa zdarzeń, 740
 - PlayerShip, 752
 - pociski, 753
 - pole walki, 748
 - rozgrywka, 743
 - Shot, 753
 - Stars, 754
 - statek gracza, 752
 - Twinkle(), 747
 - zderzenia gracza i najeźdźców, 749
 - zegar gry, 744
 - zegary animacji, 741
 - zestrzelenie statku, 752
- ## J
- Invalid arguments, 167
 - Invalidate(), 650, 651, 653
 - IronRuby, 767
 - is, 310, 313, 690
 - iteracje, 775
- ## J
- jawne rzutowanie, 166
 - język C#, 41, 42
 - język programowania, 41, 98
 - język SQL, 59
 - język XAML, 787
 - JIT, 767
 - join, 725, 730, 731
 - just-in-time, 767
- ## K
- kanały RSS, 785
 - katalogi, 448
 - lista plików, 448
 - tworzenie, 448
 - usuwanie, 448
 - katalogi projektu, 81

- KeyDown, 196, 740, 742
- KeyUp, 740, 742
- klasy, 88, 89, 91, 92, 111, 124, 125
 - [Serializable], 470
 - atrybuty, 469
 - BackgroundWorker, 768
 - BinaryFormatter, 468
 - BinaryReader, 476
 - BinaryWriter, 475
 - Bitmap, 640, 641
 - Brush, 644
 - Clone, 678
 - Console, 437
 - Control, 624
 - CryptoStream, 442
 - DataContractSerializer, 782
 - DateTime, 596
 - destruktor, 676
 - diagram klasy, 142
 - Directory, 448
 - dziedziczenie, 257, 258, 266
 - Exception, 492, 496
 - File, 448, 484
 - FileInfo, 448, 484
 - FileStream, 436
 - Font, 645
 - Form, 613, 634
 - Graphics, 640
 - GZipStream, 435
 - hermetyzacja, 229
 - hierarchia klas, 259, 264
 - implementacja interfejsu, 305, 312
 - instancja, 130
 - interfejsy, 302
 - klasy bazowe, 258, 261
 - klasy konkretne, 326
 - klasy pochodne, 425
 - klasy potomne, 261
 - konstruktory, 239, 569
 - kontrola dostępu do składowych, 224
 - LINQ to SQL, 729, 732
 - List, 364
 - MemoryStream, 435
 - MessageBox, 92
 - metody, 89
 - metody statyczne, 135
 - naturalna struktura, 142
 - nazwy, 146
 - NetworkStream, 435
 - partial, 111
 - Pen, 644
 - pola wewnętrzne, 235
 - PrintDocument, 662
 - PrintPreviewDialog, 663
 - projektowanie, 124, 142, 154
 - przeźrzenie nazw, 97
 - Queue, 405
 - Random, 186, 226
 - Rectangle, 645
 - Renderer, 616, 631
 - sealed, 700
 - składowe, 321
 - składowe prywatne, 223
 - Stack, 405
 - Stream, 434, 435
 - StreamReader, 444, 481
 - StreamWriter, 437, 440, 481
 - StringBuilder, 763
 - TimeSpan, 596
 - tworzenie, 127
 - tworzenie obiektu, 129
 - Type, 771
 - widoczność
 - składowych, 322
 - właściwości, 136
 - zasięg składowych, 322
 - zdarzenia, 534
 - klasy abstrakcyjne, 326, 328, 332
 - tworzenie, 328
 - klatki animacji, 622
 - klawiatura, 196, 742
 - klucz główny, 61, 62
 - klucze, 391, 730
 - kod C#, 79
 - kod IL, 767
 - kod SQL, 59
 - kod źródłowy, 54, 82, 88, 111
 - kodowanie, 84, 104
 - kolejki, 405
 - obsługa, 406
 - kolekcje, 355, 405, 603, 715, 716
 - Controls, 613
 - Dictionary, 391
 - enumeratory, 776
 - foreach, 367
 - indeksatory, 776
 - inicjalizatory, 372
 - klasy, 363
 - kolejki, 405, 406
 - kolekcje generyczne, 368, 371, 405
 - LINQ, 711
 - List<T>, 363
 - listy, 363, 364, 371
 - przetwarzanie, 367
 - Queue, 405
 - słowniki, 391
 - sortowanie, 374
 - Stack, 405
 - stos, 405, 407
 - tablice, 371
 - kolory, 114, 115
 - tło, 114
 - tło kontrolki, 113
 - kolumny, 60
 - komentarze, 104
 - XML, 87, 758
 - kompilacja, 74, 83
 - kompilator, 83
 - JIT, 767
 - komponenty, 446
 - FlowLayoutPanel, 451
 - TableLayoutPanel, 451
 - komunikaty
 - klawiatura, 742
 - wyjątki, 499
 - konfiguracja źródła danych, 58, 69
 - konkatenacja, 166
 - konstruktory, 239, 240, 245, 569
 - dziedziczenie, 283
 - parametry, 240
 - przeciążanie, 387
 - stosowanie, 239
 - wyjątki, 507
 - kontrola dostępu do składowych, 224
 - kontrolki, 50, 611, 612
 - animacja, 621
 - CheckBox, 72
 - ComboBox, 291
 - Controls, 613
 - DataGridView, 70
 - DateTimePicker, 457, 525
 - dodawanie
 - do formularza, 612
 - etykiety, 620
 - GroupBox, 209, 409
 - kontrolki niewizualne, 194, 446
 - Label, 113, 620
 - ListBox, 193, 395
 - NumericUpDown, 126, 214
 - obiekty, 613
 - PictureBox, 52, 86, 204, 415, 612, 621
 - podwójne buforowanie, 657
 - ProgressBar, 770
 - przerysowywanie, 650
 - przycisk, 620
 - RadioButton, 206, 409
 - StatusStrip, 193
 - TabControl, 253
 - TextBox, 126
 - Timer, 193
 - ToolStrip, 607, 614
 - TrackBar, 651
 - tworzenie, 612, 621, 626
 - UserControl, 626, 654
 - usuwanie, 612, 624, 625
 - wyświetlanie elementów wizualnych, 618
 - zdarzenia, 226
 - konwersja łańcucha znaków na tablicę bajtów, 449
 - konwersja na łańcuch znaków, 217
 - konwersja niejawną, 544
 - kopiowanie obiektów, 686
 - kopiowanie do pliku, 468
 - kowariancja, 384
 - krokowe wykonanie programu, 102

L

 - Label, 113, 456, 620
 - lambda, 780
 - Length, 184, 185
 - library assembly, 764
 - liczby, 162, 164
 - liczby całkowite, 160
 - liczby losowe, 185
 - liczby rzeczywiste, 160
 - liczby szesnastkowe, 480
 - liczby zmiennoprzecinkowe, 160
 - linie, 642, 643
 - LINQ, 567, 604, 605, 701, 710, 749
 - by, 730
 - descending, 718
 - equals, 725, 730
 - from, 718
 - group, 730
 - grupowanie wyników, 721
 - join, 725, 730, 731
 - klasy LINQ to SQL, 729, 732
 - klucze, 722, 730
 - kolekcje, 711, 716
 - łączenie kolekcji, 725
 - metody rozszerzające, 711
 - on, 725, 730
 - orderby, 718, 721
 - select, 718, 721

- select new, 729, 730
 - składnia, 712, 727
 - słowa kluczowe, 717
 - var, 730
 - where, 718
 - wyniki zapytania, 717
 - wrażenia lambda, 780, 781
 - zapytania, 711, 713, 714
 - zastosowanie, 716
 - LINQ to SQL, 732
 - LINQ to SQL Classes, 729, 730
 - LINQ to XML, 784
 - wczytywanie plików XML, 785
 - zapisywanie plików XML, 785
 - zapytania, 785
 - LINQPad, 733
 - List, 364, 371
 - List<T>, 363, 603, 709, 717
 - lista błędów, 85
 - lista kontaktowa, 60
 - ListBox, 193, 194, 387, 395, 409
 - listy, 362, 363, 367, 371
 - dynamiczna zmiana rozmiaru, 367
 - operacje, 364
 - Sort, 374
 - sortowanie, 374
 - tworzenie, 373
 - listy stałych, 371
 - listy wyliczeniowe, 357
 - literały, 161
 - Location, 87, 624
 - long, 160, 163
- Ł**
- łańcuchy strumieni, 442
 - łańcuchy znaków, 217, 251, 356, 473, 702, 763
 - Unicode, 472
 - łączenie, 166, 449, 763
 - łączenie instrukcji warunkowych, 779
 - łączenie łańcuchów znaków, 166, 449, 763
 - łączenie w łańcuchy, 537
- M**
- Main(), 90, 91, 93, 94, 277
 - managed resources, 676
 - martwy obiekt, 677
 - maszyna wirtualna, 83, 188
 - Math.Min(), 133
 - Max(), 718
 - MaximizeBox, 73
 - MeasureString(), 645
 - mechanizm oczyszczania pamięci, 179, 677, 678
 - mechanizm renderujący, 617, 628
 - MemoryStream, 435
 - Message, 514
 - MessageBox, 85, 92, 166, 547
 - Show(), 88, 99, 105, 166, 545
 - metadane, 469
 - metapoznanie, 33
 - metody, 55, 88, 89, 124, 126, 140
 - akcesory, 235, 241
 - Close(), 441
 - CompareTo(), 375
 - destrukторы, 676
 - Dispose(), 453, 625, 627, 678
 - Equals(), 772
 - finalizatory, 676
 - GetType(), 771
 - Invalidate(), 651, 653
 - konstrukторы, 239
 - kontrola dostępu, 224
 - Main(), 90, 91, 93, 94
 - metody abstrakcyjne, 326, 329
 - metody anonimowe, 780
 - metody konkretne, 326
 - metody prywatne, 226
 - metody publiczne, 226, 233, 238
 - metody rozszerzające, 700, 711
 - metody statyczne, 135
 - metody sygnalizujące zdarzenia, 557
 - metody wirtualne, 270, 275
 - modyfikatory, 692
 - nazwane argumenty, 694
 - nazwy, 141
 - parametry, 89, 167
 - parametry wyjściowe, 692
 - polimorfizm, 337
 - Print(), 662
 - przeciążanie, 385
 - przesłanianie, 262, 278
 - Read(), 435
 - ReadLine(), 441
 - refaktoryzacja, 778
 - Refresh(), 653
 - rekurencja, 691
 - return, 89, 124, 125
 - Seek(), 435
 - Show(), 635
 - Sort(), 374, 375
 - static, 135
 - ToString(), 166, 217, 358, 381, 382, 444
 - TryParse(), 693
 - typ wynikowy, 124, 125
 - ukrywanie, 278
 - Update(), 653
 - wartość zwracana, 89
 - Write(), 435, 436
 - wywołanie, 187
 - Min(), 718
 - MinimizeBox, 73
 - model klas, 338
 - modyfikacja kodu, 87
 - modyfikatory, 692
 - out, 692
 - ref, 693
 - modyfikatory dostępu, 306, 321, 322, 766
 - internal, 321, 766
 - private, 321
 - protected, 321
 - public, 321, 766, 767
 - sealed, 321
 - Mono, 767
 - MouseClick, 630
 - mózg, 31
 - MultiColumn, 193
 - myślenie, 31
- N**
- nadawcy zdarzeń, 548
 - Name, 194, 386
 - namespace, 197
 - nasłuchiwanie zdarzeń, 531
 - naturalne klasy, 142
 - nawiasy, 89
 - nawiasy klamrowe, 94, 103, 251
 - nazwane argumenty, 694
 - nazwy, 243, 370, 764
 - interfejsy, 303
 - klasy, 140, 146, 154
 - metody, 141, 154
 - pliki, 51
 - przestrzenie nazw, 111
 - zmiennne, 98, 172
 - NetworkStream, 435
 - new, 128, 130, 132, 148, 152, 197, 279, 281, 308
 - New Project, 48
 - Next(), 359
 - niejawne konwersje, 544
 - nieobsłużony wyjątek, 488, 500, 510
 - notacja Pascal, 234
 - notacja wielbłądzia, 234
 - null, 188, 316, 695, 761
 - nullable type, 695
 - Nullable<T>, 695
 - NullReferenceException, 152, 536, 557
 - NumericUpDown, 126, 161, 214, 215, 254, 535
 - nvarchar, 63
- O**
- obiekt porównujący, 377
 - obiekty, 121, 128, 130, 173, 308, 685, 687
 - akcesory, 235
 - deserializacja, 464, 465
 - destrukторы, 676
 - finalizatory, 676
 - formularze, 187
 - inicjalizacja, 153, 239
 - konstrukторы, 239
 - kontrolki, 613
 - metody, 55, 88, 89, 124, 126, 140
 - obiekt martwy, 677
 - poła, 136
 - polimorfizm, 337
 - przechowywanie informacji, 136
 - referencje, 174, 175, 187, 189
 - serializacja, 464, 465
 - składowe, 321
 - stan, 136, 465, 466
 - sterta, 138
 - this, 187
 - tworzenie, 128, 131
 - usuwanie z pamięci, 176, 179
 - właściwości, 136
 - zdarzenia, 533
 - zmiana danych, 182
 - zmiennne, 173
 - obiekty bazy danych, 46
 - obiekty wizualne, 46, 57
 - object, 161
 - Object, 471
 - Equals(), 772

- Object
 - ReferenceEquals(), 772
 - Object Relational Designer, 729, 730
 - obliczenia na kolekcjach, 716
 - obrazy, 52
 - rysowanie, 644
 - zasoby, 641
 - obsługa danych, 605
 - obsługa formularza, 254
 - obsługa klawiszy, 196
 - obsługa wyjątków, 487, 503, 522, 523, 527, 607
 - wiele bloków catch, 514
 - zalecenia, 524
 - obsługa zdarzeń, 215, 255, 531, 533
 - oczyszczanie pamięci, 176, 179, 188, 678, 683
 - odbiorcy zdarzeń, 548
 - odczytywanie
 - bajty ze strumienia, 482
 - dane, 434
 - dane binarne, 476
 - pliki, 441
 - pliki XML, 785
 - odporność na błędy, 502
 - odporność programów, 696
 - odświeżanie pól tekstowych, 151
 - odwracanie kolejności
 - bajtów, 474
 - okienko z komunikatem, 93
 - okna dialogowe, 445, 446, 449
 - DialogResult, 445
 - OpenFileDialog, 446
 - otwieranie plików, 447
 - PrintDialog, 663
 - SaveFileDialog, 447
 - ShowDialog(), 445
 - wyświetlanie, 445
 - zapisywanie plików, 447
 - okrąg, 642, 643
 - on, 725, 730
 - OnClick, 557
 - OnDragOver, 537
 - OnPaint(), 653
 - OOM, 336
 - opakowanie, 690
 - OpenFileDialog, 446, 451, 627
 - OpenRead(), 448
 - OpenWrite(), 448, 477
 - operatory, 100, 105, 172
 - ?, 761
 - ??, 761
 - ++, 761
 - =, 761
 - ==, 772
 - dwukropek, 304
 - operatory logiczne, 106
 - operatory warunkowe, 106
 - przeciążanie, 774
 - OR, 762
 - orderby, 717, 718, 721
 - organizacja klas, 144
 - otwieranie plików, 437, 451
 - out, 692
 - Output, 236
 - OverflowException, 496
 - override, 270, 276, 280, 281, 381
- ## P
- paczka instalatora, 47
 - PadLeft(), 762
 - Paint, 650, 651, 653, 745
 - podwójne buforowanie, 657
 - PaintEventArgs, 651
 - pamięć, 689
 - pamięć bazy danych, 47
 - parametry, 89, 167
 - parametry opcjonalne, 694
 - parametry wyjściowe, 692
 - przekazywanie przez referencję, 693
 - przekazywanie przez wartość, 693
 - typy danych, 167
 - wartości domyślne, 694
 - parametry typu, 368
 - Parse(), 490, 525, 693
 - partial, 97, 111, 767
 - Pascal, 234
 - pasek tytułowy, 73
 - PATH, 728
 - Pen, 644
 - pętle, 103, 107, 111, 115
 - break, 760
 - continue, 760
 - for, 103, 107
 - foreach, 367, 382
 - pętle nieskończone, 109
 - while, 103, 107
 - PictureBox, 52, 53, 54, 86, 204, 206, 208, 415, 429, 430, 612, 617, 621, 622, 632
 - pisanie kodu, 42, 79, 85, 132
 - platforma .NET, 82
 - pliki, 433, 436
 - EndOfStream, 441
 - File, 448
 - informacje o pliku, 448
 - odczytywanie, 441
 - okna dialogowe, 451
 - otwieranie, 437
 - przechowywanie
 - obiektów, 464
 - sprawdzanie istnienia, 448
 - StreamWriter, 437
 - tworzenie, 437, 478
 - XML, 478, 785
 - zamykanie, 437
 - zapisywanie, 436, 437, 460
 - pliki binarne, 472, 479
 - obsługa, 479
 - wczytywanie, 476
 - widok szesnastkowy, 480
 - zapisywanie, 472, 475
 - pliki graficzne, 638
 - pliki projektu, 56, 81
 - pliki rozwiązania, 82
 - pliki wykonywalne, 56, 83
 - pliki zasobów, 623
 - pobieranie danych, 710
 - podajmowanie decyzji, 105
 - podgląd wartości zmiennych, 101
 - podgląd wydruku, 663, 664
 - podklasy, 258
 - podwójne buforowanie, 656
 - DoubleBuffered, 657, 659
 - Paint, 657
 - Point, 698
 - pola, 136, 148
 - kontrola dostępu, 224
 - pola prywatne, 223, 226
 - pola publiczne, 233
 - pola wewnętrzne, 235
 - pole tekstowe, 72, 150
 - odświeżanie, 151
 - pole wyboru, 113
 - polimorfizm, 336, 337, 363
 - połączenie formularza z bazą danych, 57, 68
 - porównywanie
 - liczby, 106
 - obiektów, 375, 378
 - pliki binarne, 477
 - potrzeby użytkowników, 45
 - powiązanie kontrolki z bazą danych, 70
 - primary key, 61
 - Print(), 662, 663
 - PrintDialog, 663
 - PrintDocument, 662, 663
 - PrintPage, 664
 - PrintPreviewDialog, 663
 - PrintTableRow(), 664
 - private, 223, 225, 226, 232, 238, 306, 321, 324, 558
 - procedura oczyszczania pamięci, 176
 - procedury obsługi zdarzenia, 531, 533, 535, 538, 539, 607
 - procedury składowane, 59
 - proces wdrażania aplikacji, 75
 - process assembly, 764
 - program, 46, 82, 91
 - argumenty wywołania, 482
 - budowa, 88
 - punkt wejścia, 90
 - program instalacyjny, 76
 - programowanie, 84, 232
 - programowanie obiektowe, 336, 585
 - ProgressBar, 770
 - ProgressChanged, 768
 - projekt, 48, 51, 149
 - budowanie, 74
 - Console Application, 276
 - katalogi, 81
 - pliki, 56, 81
 - Windows Forms Application, 80, 86
 - WPF Application, 786
 - projektowanie formularze, 86
 - klasy, 124, 142, 154
 - Properties, 72, 194, 622
 - Properties.Resources, 623
 - prostokąty, 642
 - protected, 321, 324
 - prywatne pola, 223
 - prywatne składowe, 226
 - przechowywanie
 - dane, 355
 - obiektów w pamięci, 138
 - obiektów w plikach, 464
 - stan obiektu, 136
 - tekst, 473
 - wartości, 98
 - przechwytywanie wyjątków, 155
 - przeciążanie
 - konstruktory, 387, 456
 - metody, 385
 - operatory, 774
 - przekazywanie przez referencję, 693
 - przekazywanie przez wartość, 693
 - zapełnienie, 165

- przerysowanie formularza, 347
 przerysowanie kontrolek, 650
 przesłanianie, 243, 262, 270, 280
 przestrzenie nazw, 82, 88, 91, 97, 111, 764
 przesunięcie bitowe, 763
 przetwarzanie kolekcji, 367
 przezroczystość, 637, 649
 przyciski, 150, 171, 620
 przyciski maksymalizacji i minimalizacji, 193
 przypisanie referencji, 187
 przypisanie wartości, 99, 105
 public, 197, 223, 225, 233, 321, 324, 766, 767
 Publish Wizard, 75
 pułapki, 500
 punkt przerwania, 101, 375
 punkt wejścia programu, 90, 91, 92, 275
- ## Q
- Queue, 405
- ## R
- RadioButton, 206, 409
 Random, 186, 195, 226, 581
 Next(), 359
 Read(), 435
 ReadAllBytes(), 473, 474, 477
 ReadAllText(), 451, 484
 ReadBlock(), 481
 ReadBytes(), 476
 ReadChar(), 476
 ReadInt32(), 476
 ReadKey(), 375, 763
 ReadLine(), 441
 ReadObject(), 783
 ReadOnly, 395
 ReadSingle(), 476
 ReadString(), 476
 Rectangle, 645
 ref, 693
 Refactor/Extract Method, 778, 779
 Refactor/Rename, 779
 refaktoryzacja, 778
 instrukcje warunkowe, 779
 metody, 778
 nazwy zmiennych, 779
 referencje, 174, 175, 179, 182, 189, 275, 279, 466, 686
 efekty uboczne, 177
 null, 188
 przekazywanie parametrów, 693
 referencje interfejsów, 308, 309, 324
 referencje obiektów, 324
 referencje wielokrotne, 177
 Refresh(), 653
 rekurencja, 691
 Remove(), 364, 368, 392, 618, 716
 RemoveAt(), 368
 Render(), 617
 Renderer, 616, 628, 631, 657
 ResizeImage(), 639
 renderowanie grafiki, 616
 ReportProgress(), 769, 770
 Reset Window Layout, 51
 Resize, 639
 ResizeImage(), 639, 640, 654
 Resources.resx, 622
 return, 89, 124, 125
 Reverse(), 474
 rozpowszechnianie aplikacji, 76
 rozszerzanie klas, 265
 rozszerzanie typu podstawowego, 702
 rozwiązanie, 82
 rozwiązywanie problemów, 154
 różność, 772
 RSS, 785
 Run, 74
 Run Worker Async(), 769
 Run Worker Completed, 768, 770
 rysowanie, 616, 628, 642, 654
 bitmapy, 641
 Brush, 644
 elipsy, 643
 formularze, 653
 Graphics, 644
 Invalidate(), 650, 651
 kolejność rysowania, 644
 kontrolki, 653
 linie, 643
 obrazy, 644
 okręgi, 643
 Paint, 650, 653
 Pen, 644
 pióro, 644
 podwójne buforowanie, 656
 problemy wydajnościowe, 637
 przerysowywanie, 650
 przezroczystość, 649
 Refresh(), 653
 tekst, 643, 645
 Update(), 653
 współrzędne, 643
 rzutowanie typów, 164
 liczby na typ wylczeniowy, 358
 rzutowanie w dół, 316
 rzutowanie w górę, 315, 318, 384
 typ wylczeniowy na liczbę, 358
- ## S
- Save, 48
 Save All, 48, 67
 Save(), 457
 SaveFileDialog, 447, 451
 sbyte, 160
 Scroll, 652
 sealed, 321, 700, 701
 Seek(), 435
 sekwencje formatujące, 104
 select, 717, 718, 721
 SELECT, 46
 select new, 729, 730, 731
 Select Resource, 53, 54
 serializacja, 464, 465, 468, 470, 477, 485, 607, 680
 [Serializable], 469, 470
 BinaryFormatter, 468
 DataContractSerializer, 782
 Dispose(), 681
 SerializationException, 501, 502, 508, 606, 607
 Serialize(), 468
 Server Explorer, 58
 set, 235, 237, 241, 245
 Set as Startup Project, 767
 short, 160, 163
 Show Data Sources, 70
 Show Next Statement, 497
 Show(), 635
 ShowDialog(), 445, 451
 DialogResult, 445
 Size, 87, 645, 651
 skalowanie obrazków, 640
 składnia LINQ, 712, 727
 składowe, 321
 składowe prywatne, 223
 słowa kluczowe, 168, 197
 abstract, 328, 329
 as, 313, 690
 base, 282
 break, 461, 760
 case, 461
 catch, 503
 const, 214
 continue, 760
 delegate, 549, 550
 else, 105
 event, 534, 558
 finally, 508, 509
 for, 103, 107
 foreach, 367, 382
 goto, 761
 if, 105, 167, 460
 interface, 303
 internal, 321, 766
 is, 310, 313, 690
 new, 128, 279
 null, 188
 override, 270, 276, 281, 381
 partial, 97, 111
 private, 223, 225, 232, 321
 protected, 321
 public, 223, 225, 321, 766
 ref, 693
 return, 89, 124
 sealed, 321, 700, 701
 static, 135
 struct, 685
 switch, 461
 this, 187, 189
 try, 503
 typeof, 771
 using, 88, 454
 var, 730
 virtual, 270, 276, 281
 void, 124
 while, 107
 yield, 775
 słowa zarezerwowane, 168
 słowniki, 391, 628
 Dictionary, 391
 dodawanie elementu, 392
 klucze, 391, 392
 obsługa, 393
 określanie liczby par, 392
 pobieranie listy kluczy, 392
 usuwanie elementu, 392
 wartości, 391, 392
 wyszukiwanie wartości, 392
 snippets, 85
 Solution Explorer, 51, 54, 58, 82, 622
 Solution Manager, 149
 Sort(), 374, 375
 sortowanie, 374, 376, 379
 sortowanie obiektów, 377

- specyfikacja, 394
- Split(), 463
- sprawdzanie implementacji interfejsu, 310
- sprawdzanie wartości zmiennych, 106
- sprawdzanie warunków, 106
- SQL, 58, 59, 715, 732, 733
 - instrukcje, 59
 - procedury składowane, 59
 - zapytania, 59
- SQL Database, 58
- SQL Server Compact Edition, 43, 731
- SqlMetal.exe, 728
- Stack, 405
- StackTrace, 496
- stałe, 214, 568
- stan obiektu, 136, 465, 466
- standardowa procedura obsługi zdarzenia, 537
- standardowe okna dialogowe, 445
- Start Debugging, 83, 94, 102
- static, 135
- StatusStrip, 193, 194
 - konfiguracja, 194
- Step Into, 498, 506, 551, 776
- Step Over, 102, 497, 498, 506
- sterta, 138, 689
- Stop, 497
- Stop Debugging, 56, 115
- stos, 405, 407, 689, 691, 698
 - kładzenie elementów, 407
 - obsługa, 407
 - podnoszenie elementów, 407
- stos wywołań, 501
- Stream, 434, 441
 - Read(), 482
- StreamReader, 441, 444, 449, 481, 484
 - ReadBlock(), 481
- StreamWriter, 437, 438, 439, 440, 444, 448, 449, 481, 484
- string, 160, 163, 473, 702
- String.Format(), 480, 593, 718
- String.IsNullOrEmpty(), 290
- String.PadLeft(), 762
- StringBuilder, 763
- struct, 685
- struktura programu, 46
- struktury, 685, 687, 698
- strumienie, 434, 442, 468, 682
 - Dispose(), 454
 - FileStream, 435, 436
 - GZipStream, 435
 - łańcuchy strumieni, 442
 - MemoryStream, 435
 - NetworkStream, 435
 - odczytywanie bajtów, 482
 - odczytywanie danych, 435
 - przesyłanie danych między strumieniami, 442
 - Stream, 434
 - StreamReader, 444, 481
 - StreamWriter, 437, 481
 - zamykanie, 449, 454
 - zapisywanie danych, 435
 - zmiana położenia wewnątrz strumienia, 435
- style nauczania, 34
- subskrypcja zdarzeń, 532, 535, 553
- Substring(), 251, 481
- Sum(), 718
- switch, 461, 462
 - break, 461
 - case, 461
 - default, 461, 462
- sygnatura zdarzenia, 537
- symbol łamania wiersza tekstu, 55
- symbole matematyczne, 100
- symulator ula, 565, 614
- symulator wyścigów, 202
 - architektura aplikacji, 206
 - Bet, 207
 - formularz, 204
 - Greyhound, 204, 208
 - Guy, 205
 - interfejs użytkownika, 208
 - klasy, 204
 - obstawianie, 203
 - wyścig, 203
 - zawieranie zakładów, 209
- system nawigacyjny, 123
- system szesnastkowy, 480
- system turowy, 412, 583
- system zarządzania ulem, 289
- System.Collections.Generic, 405
- System.Drawing, 570, 633, 642
- System.IO, 438
- System.Linq, 710
- System.Windows.Forms, 82, 88, 125, 765
- System.Xml.Linq, 784
- S**
- śledzenie przepływu w blokach try/catch, 506
- śledzenie wartości zmiennych, 101
- średniki, 104
- środowisko programistyczne, 84
- T**
- TabControl, 253
- tabele, 60, 65, 715
 - automatyczne generowanie identyfikatorów, 61
 - klucz główny, 61
 - kolumna ContactID, 60
 - kolumny, 60, 62
 - tworzenie, 60
 - typy danych, 60
- TableLayoutPanel, 451
- tablica metod wirtualnych, 275
- tablice, 183, 184, 361, 362, 371
 - długość, 184
 - elementy, 183
 - indeks, 183
 - Length, 184
 - odwołanie do elementów, 184
 - tworzenie, 183
 - wartości elementów, 183
- Take(), 718
- technologia LINQ, 605
- tekst, 473, 645
- testowanie, 77
- testy logiczne, 761
- Text, 72, 73, 113, 624
- TextBox, 126, 251, 395, 451
- TextChanged, 255
- TextReader, 441
- this, 187, 189, 240, 537
- Tick, 594, 595, 741
- Timer, 193, 194, 594
 - konfiguracja, 194
- TimeSpan, 596
- Title, 451
- to formularza, 639
- ToArray(), 717
- ToDateTime(), 457
- ToDictionary(), 717
- Toggle Breakpoint, 101, 498
- ToInt32(), 762
- ToList(), 717
- Toolbox, 49, 50, 52, 557, 612, 624
 - dodawanie kontroltek, 621, 626
- ToolStrip, 607, 614
- ToString(), 166, 217, 358, 381, 382, 383, 444, 457
- TrackBar, 651, 652
- true, 100, 160
- try, 503, 505, 519
- tryb Basic Settings, 497
- tryb Expert, 497
- tryb szesnastkowy, 497
- TryParse(), 693, 696
- tworzenie
 - aplikacje, 43
 - aplikacje Windows, 80
 - aplikacje WPF, 786
 - baza danych, 58
 - diagram klasy, 142
 - enumeratory, 776
 - hierarchia klas, 264
 - instancja, 131, 174
 - interfejs użytkownika, 52
 - interfejsy, 303
 - katalogi, 448
 - klasy, 92, 127
 - klasy abstrakcyjne, 328
 - kontrolki, 612, 621, 626
 - kopie obiektów, 686
 - listy, 373
 - łańcuchy strumieni, 442
 - objekty, 128, 131, 161
 - pgtę, 103
 - pliki, 437, 471, 478
 - program, 82
 - projekt, 48, 86, 149
 - tabele, 60
 - tablice, 183
 - wyjątki, 516
 - zmiennie referencyjne, 174
- typ wynikowy metody, 124
- Type, 771
- typeof, 771
- typy akceptujące wartości puste, 695, 696
- typy anonimowe, 725, 731, 780
- typy danych, 98, 99, 159, 160, 189
 - bool, 160
 - char, 161
 - decimal, 161
 - double, 160
 - enum, 357
 - float, 160, 161
 - int, 160, 163
 - liczby całkowite, 160

- object, 161
 - rzutowanie, 164
 - short, 163
 - string, 160
 - typy generyczne, 368
 - typy ogólne, 363
 - typy parametrów, 167
 - typy referencyjne, 687
 - typy wartościowe, 189, 371, 687
 - typy wyliczeniowe, 355, 357, 358
- U**
- uchwyty okien, 682
 - uczenie się, 34, 35
 - uint, 160
 - ukrywanie informacji, 221, 229
 - ulong, 160
 - Unicode, 472, 473, 483, 484
 - unikanie powielania kodu, 304
 - UPDATE, 46
 - Update(), 653
 - uruchamianie aplikacji, 56, 74
 - UserControl, 626, 654
 - ushort, 160
 - using, 82, 88, 92, 197, 454, 519, 521, 570
 - usuwanie
 - elementy kolekcji, 585
 - katalogi, 448
 - kontrolki, 612, 625
 - obiekty, 176
 - UTF-8, 484
 - użytkownicy programu, 45
- V**
- Value, 161
 - ValueChanged, 535
 - var, 730
 - VB.NET, 767
 - virtual, 270, 275, 276, 280, 281, 294
 - Visible, 117, 415, 624
 - Visual Basic.NET, 767
 - Visual C#, 78
 - Visual Studio 2010, 36, 41, 42, 43, 48
 - budowanie projektu, 74
 - Database Explorer, 58
 - Debug, 56
 - debugowanie, 56
 - edycja formularza, 55
 - Error List, 49, 50, 85
 - Form Designer, 86
 - interfejs użytkownika, 50
 - konfiguracja źródła danych, 58
 - nazwy plików, 51
 - pasek narzędzi, 49
 - Properties, 72
 - Publish Wizard, 75
 - Reset Window Layout, 51
 - Select Resource, 53, 54
 - Server Explorer, 58
 - snippets, 85
 - Solution Explorer, 51, 54, 58
 - Toolbox, 49, 50, 52
 - tworzenie projektu, 48
 - uruchamianie aplikacji, 56
 - Visual Studio 2010 Express Edition, 36, 48
 - void, 124, 151
 - vtable, 275
- W**
- wartości, 391, 686
 - binarne, 161
 - logiczne, 160
 - null, 188, 695
 - szesnastkowe, 480
 - wartości domyślne, 694
 - wartości puste, 695
 - wartość wynikowa metody, 124
 - warunki, 106
 - Watch, 102, 499, 500, 505
 - wątki, 768
 - wbudowane kolekcje generyczne, 405
 - wczytywanie
 - dane binarne, 476
 - dane z kanału RSS, 785
 - pliki XML, 785
 - wdrażanie aplikacji, 75
 - where, 717, 718
 - while, 107, 111, 115, 197
 - wiązanie danych, 70
 - widoczność składowych, 322
 - widok szesnastkowy, 479, 480
 - Width, 624
 - wielkość liter, 243
 - wielokrotne instrukcje
 - using, 454
 - wielokrotne wykorzystanie kodu, 261
 - wiersz polecenia, 482
 - Windows Communication Foundation, 782
 - Windows Form Designer, 87
 - Windows Forms Application, 80, 82, 86, 101
 - Windows Presentation Foundation, 786
 - właściwości, 136, 235, 241, 245
 - automatyczne, 237, 568
 - kontrolki, 72
 - prywatne, 236
 - tylko do odczytu, 571
 - WorkerReportsProgress, 770
 - WorkerSupports Cancellation, 770
 - WPF, 786, 787
 - WPF Application, 786
 - wprowadzanie kodu, 55
 - Write(), 435, 436, 437, 449, 473
 - WriteAllBytes(), 473, 474, 482
 - WriteAllText(), 451, 484
 - WriteLine(), 236, 437, 449
 - wstawianie
 - dane do bazy danych, 66
 - kontrolki, 86
 - punkt przzerwania, 101
 - wybór opcji, 461
 - wybór źródła danych, 70
 - wyciąganie metod, 778
 - wydajność, 638
 - wygląd aplikacji, 73
 - wyjątki, 489, 491, 492, 493, 495, 510, 587
 - ArgumentNullException
 - Exception, 516
 - blok catch, 503, 505, 510, 522
 - blok finally, 508, 509
 - blok try, 503
 - DivideByZero
 - Exception, 491
 - Exception, 492, 496, 510, 513, 519
 - FileNotFoundException
 - Exception, 510
 - finalizatory, 683
 - IndexOutOfRangeException
 - Exception, 492
 - informacje o wyjątku, 513
 - kilka bloków catch, 510
 - komunikaty, 499
 - konstruktory, 507
 - Message, 514
 - nieobsłużony wyjątek, 500, 510
 - NullReference
 - Exception, 536
 - obiekty, 493
 - obsługa, 503, 524
 - przechwytywanie, 515
 - SerializationException, 501, 502, 508, 606
 - śledzenie przepływu
 - w blokach try/catch, 506
 - tworzenie wyjątków, 516
 - unikanie wyjątków, 520
 - wiele bloków catch, 514
 - wyłapywanie, 503
 - wyłapywanie różnych typów wyjątków, 514
 - zgłaszanie, 515
 - wyliczenia, 356
 - wyłapywanie wyjątków, 503
 - wymagania użytkowników, 45
 - wyniki zapytania LINQ, 717
 - Wyprawa, 411
 - architektura, 416
 - atak, 417
 - Attack(), 423
 - broń, 426, 427
 - Direction, 420
 - Enemy, 420, 424
 - formularz, 414, 429
 - Game, 416, 417, 418
 - informacje o graczku, 422
 - IPotion, 428
 - mikstury magiczne, 428
 - Move(), 423
 - Mover, 420, 421
 - okno statystyk, 415
 - Player, 420, 422
 - poruszanie się, 420
 - poziomy, 419
 - Rectangle, 429
 - system turowy, 412
 - UpdateCharacters(), 430
 - użycie obiektów, 416
 - Weapon, 426
 - wrogowie, 424
 - wyrażenia lambda, 780, 781
 - wyrównanie kontrolek, 72
 - wyświetlanie komunikatu, 104
 - wyświetlanie okien dialogowych, 445
 - wywołanie metody, 187, 265, 691
 - ukryte metody, 279
 - wywołanie zdarzeń, 532
 - wzorce projektowe, 558

- X**
 XAML, 786, 787
 XDocument, 785
 Load(), 785
 XML, 87, 478, 604, 784
 XmlDictionaryReader, 783
 XOR, 762
- Y**
 yield return, 775, 776
- Z**
 zachowanie obiektu, 136
 zadania, 37
 zakładki, 84
 zamykanie
 plik, 437
 strumień, 449, 454
 zapisywanie
 dane, 434
 dane binarne, 472, 475
 plik, 436, 449, 451, 460
 plik XML, 785
 projekt, 48
- zapytania LINQ, 604, 711, 713,
 714, 715
 LINQ to XML, 785
 zapytania SQL, 59, 715
 zarządzanie bazami danych, 43
 zarządzanie pamięcią, 689
 zasady programowania
 obiektowego, 336
 zasięg składowych, 322
 zasoby, 623
 obrazy, 641
 zasoby zarządzane, 676
 zdarzenia, 215, 226, 531
 argumenty, 534
 Click, 546, 644, 649
 delegaty, 548, 558
 EventArgs, 534
 EventHandler, 534, 537, 544
 formularze, 545
 funkcje zwrotne, 554, 558
 generowanie, 531
 inicjowanie, 532
 KeyDown, 742
 KeyUp, 742
 łączenie w łańcuchy, 537
 metody sygnalizujące
 zdarzenia, 557
 MouseClicked, 630
 nadawcy, 548
 nasłuchiwanie, 531
 obsługa, 255, 533
 odbiorcy, 548
 OnClick, 557
 Paint, 650, 653, 745
 PrintPage, 664
 procedury obsługi, 531,
 535, 538, 539
 Resize, 639
 Scroll, 652
 standardowa procedura
 obsługi, 537
 subskrypcja, 532, 535, 553
 sygnatura zdarzenia, 537
 TextChanged, 255
 Tick, 594, 741
 ValueChanged, 535
 wywoływanie, 532
 zdarzenia wielokrotnie, 594
 zegary, 594
 zdarzenia, 595
 zgłaszanie wyjątków, 492,
 515, 587
 złożenia, 764
 złożenia biblioteczne, 764
 złożenia procesów, 764
- zmiana
 dane obiektu, 182
 nazwy zmiennych, 779
 rozmiar bitmapy, 639, 640
 zmiennie, 98, 150, 162
 deklaracja, 98
 nazwa, 172
 przypisanie wartości, 99
 typ danych, 99, 160
 wartość, 98
 zmiennie lokalne, 152
 zmiennie referencyjne, 174,
 308, 548
 znacznik kolejności bajtów, 484
 znak nowego wiersza, 401
 znaki, 161, 472
 Unicode, 484
 zwalnianie zasobów, 453, 676,
 682
 zwiększanie funkcjonalności
 klas, 700
- Ź**
 źródło danych, 68, 707, 710
 konfiguracja, 69

Zobacz, jakie możliwości kryje język C#. To nie jest trudne!

Rusz głową! C#

C# to jeden z języków, dzięki którym możesz pisać przenośny kod. Nie musisz się martwić o to, jakiego systemu używa Twój klient. Najważniejsze, żeby posiadał środowisko uruchomieniowe: .NET Framework, Mono lub DotGNU. Czy nie marzyłeś zawsze o tym, żeby napisać kod raz, a potem bez żadnych dodatkowych nakładów uruchamiać go na różnych platformach? Twoje marzenia właśnie się spełniają!

Dzięki tej książce, należącej do cenniejszej serii „Rusz głową!”, opanujesz język C# w mgnieniu oka! Tylko kilkaset stron dzieli Cię od swobodnego poruszania się w kodzie napisanym w tym języku. Każda z tych stron charakteryzuje się odpowiednią dawką humoru, doskonałą przejrzystością oraz perfekcyjnie przekazaną wiedzą. Czego się nauczysz? Przede wszystkim dowiesz się, jak stworzyć działający program w 10 minut. Następnie poznasz elementy programowania obiektowego — takie pojęcia, jak hermetyzacja czy dziedziczenie nie będą Ci obce! Kolejne strony przyniosą szeroką wiedzę z zakresu operacji na plikach, obsługi wyjątków oraz tworzenia interfejsu użytkownika. Wreszcie poznasz język LINQ służący do efektywnego operowania na zbiorach danych. „C#. Rusz głową!” to idealna propozycja dla wszystkich czytelników chcących rozpocząć przygodę z językiem C# oraz platformą .NET.

- ▼ Przygotowanie środowiska pracy, zapoznanie z Visual Studio
- ▼ Wsparcie Visual Studio dla programisty
- ▼ Anatomia programu
- ▼ Praca z debuggerem
- ▼ Pętle, instrukcje warunkowe
- ▼ Elementy programowania obiektowego
- ▼ Typy zmiennych
- ▼ Referencje
- ▼ Tablice
- ▼ Hermetyzacja obiektów
- ▼ Implementacja interfejsów
- ▼ Typy wyczerpieniowe
- ▼ Operowanie strumieniami danych
- ▼ Obsługa wyjątków
- ▼ Wykorzystanie języka LINQ do operacji na bazach danych i dużych zbiorach informacji
- ▼ Tworzenie interfejsu użytkownika

helion.pl
księgarnia
internetowa



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. Kosciuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

Nr katalogowy: 5873



Księgarnia internetowa:

<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900

Cena 99,00 zł

ISBN 978-83-246-2953-4



9 788324 629534



Informatyka w najlepszym wydaniu