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 Obiektowie

Przewodnik do nauki praktycznego programowania w C# i .NET

Poznaj sekrety abstrakcji i dziedziczenia

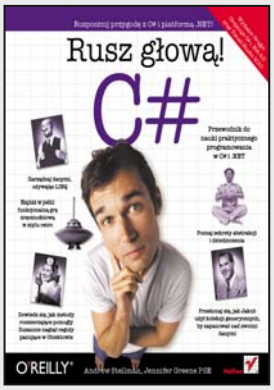
Przekonaj się, jak Jakub użył kolekcji generycznych, by zapanować nad swoimi danymi

O'REILLY®

Andrew Stellman, Jennifer Greene PSE

Helion

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 wyliczeniowe
- 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!

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

Spis treści (skrócony)

Spis treści (z prawdziwego zdarzenia)

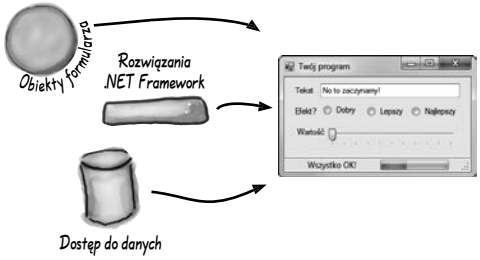
Wstęp

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#?

Zwiększ wydajność przy pomocy C#

1 Aplikacje Visual Studio w 10 minut lub mniej

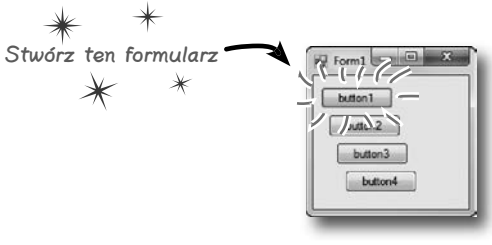
Czy chcesz tworzyć wspaniałe programy naprawdę szybko? Wraz z C# dostajesz do ręki **potężny język programowania** i wartościowe narzędzie. Dzięki **Visual Studio IDE** do historii przejdą sytuacje, w których musiałeś pisać jakiś nędzny kod, by ponownie zapewnić prawidłowe działanie przycisku. I to nie 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.



To tylko kod

2 Pod maską

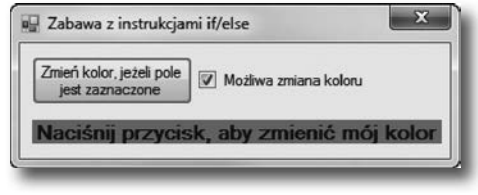
Jesteś programistą, nie jedynie użytkownikiem IDE. IDE może wykonać za Ciebie wiele pracy, ale na razie jest to wszystko, co może dla Ciebie zrobić. Oczywiście, istnieje wiele **powtarzalnych czynności** podczas pisania aplikacji i IDE okazuje się tu bardzo pomocne. Praca z nim to jednak *dopiero początek*. 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 tworzysz 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 (chociaż istnieją także bardzo małe aplikacje składające się z tylko jednej klasy).

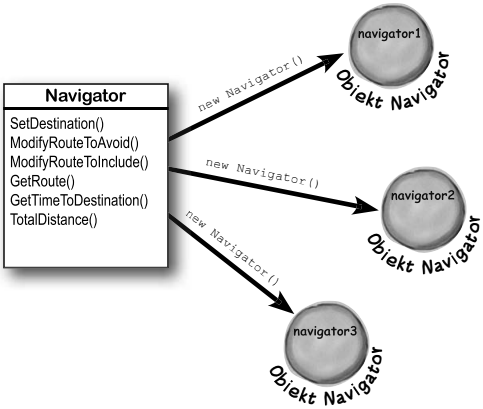
Klasa posiada jedną lub więcej metod. Twoje metody zawsze będą umieszczane **wewnątrz klas**, a każda z nich będzie się składała z instrukcji i wyrażeń — jak te, które do tej pory widziałeś.

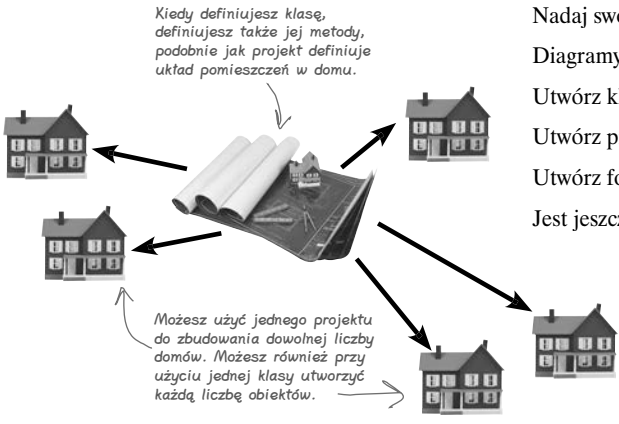


Obiekty: zorientuj się!

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 ma mechanice tworzenia kodu. Prawidłowe użycie obiektów spowoduje, że proces pisania kodu stanie się bardziej *intuicyjny*, a jego późniejsza analiza i modyfikacja — znacznie łatwiejsze.





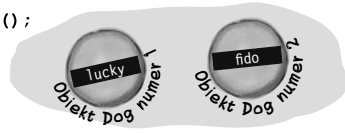
Typy i referencje

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

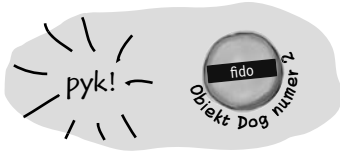
Typ danych, baza danych, dane komandora porucznika… wszystko to są ważne rzeczy. Bez danych Twoje programy są bezużyteczne. Potrzebujesz informacji dostarczanych przez użytkowników. Na jej podstawie wyszukujesz lub tworzysz nową **informację** i zwracasz ją użytkownikom. W rzeczywistości prawie wszystko, 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** (*psssst… obiekty to także dane*).

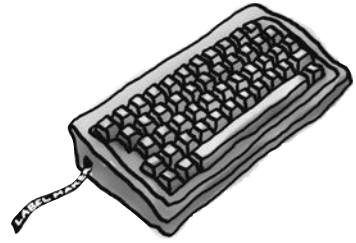

```
Dog fido;
Dog lucky = new Dog();
```


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




```
lucky = null;
```

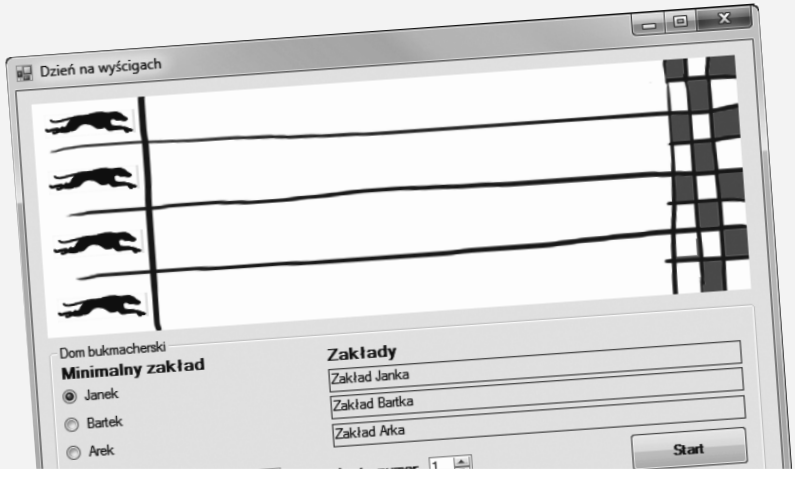




Laboratorium C# numer 1

Dzień na wyścigach

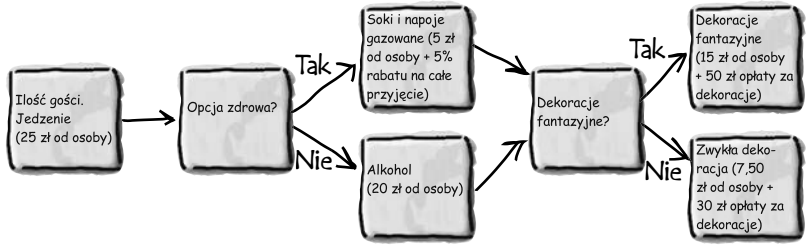
Janek, Bartek i Arek uwielbiają chodzić na tor wyścigowy, ale ciągła utrata pieniędzy powoduje u nich frustrację. Potrzebują symulatora, aby mogli określić zwycięzcę, zanim wyłożą pieniądze na zakłady. Jeśli dobrze wywiążesz się z zadania, będziesz miał procenty z ich wygranych.



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**.



Dziedziczenie

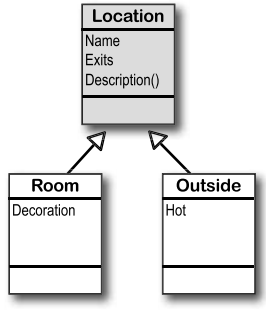
6 Drzewo genealogiczne Twoich obiektów

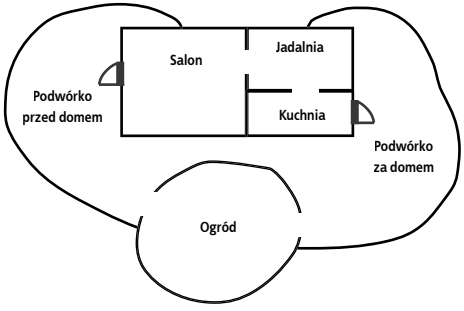
Czasami CHCIAŁBYŚ być dokładnie taki sam jak Twoi rodzice. Czy kiedykolwiek natknąłeś się na obiekt, który robiłby ***prawie*** wszystko, czego byś sobie od niego życzył? Czy kiedykolwiek znalazłeś się w takiej sytuacji, że gdybyś ***zmienił dosłownie kilka rzeczy***, obiekt byłby doskonały? Cóż, to tylko jeden z wielu powodów, które sprawiają, że **dziedziczenie** zalicza się do najważniejszy 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**.

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?

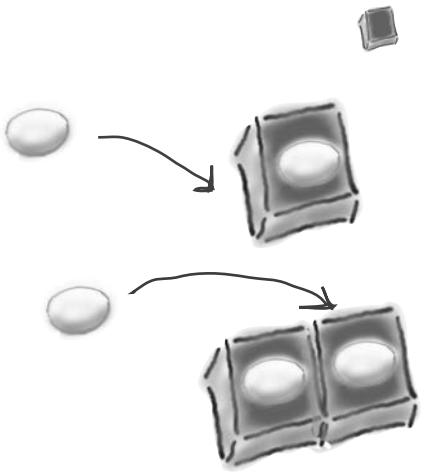


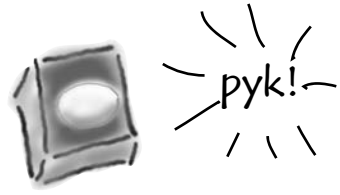


Typy wyliczeniowe i kolekcje

8 Przechowywanie dużej ilości danych

Z deszczu pod rynnę. 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.

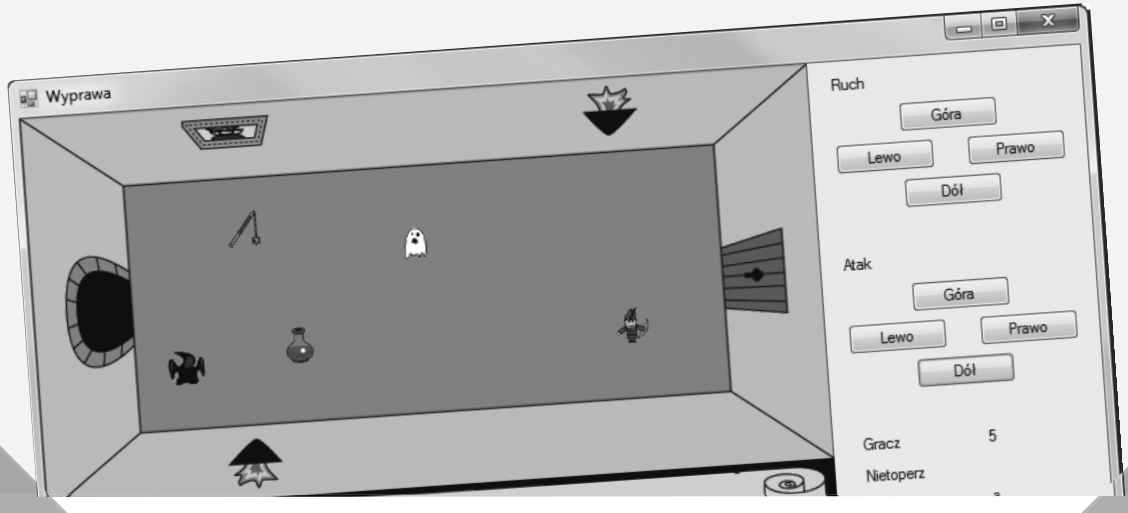




Laboratorium C# numer 2

Wyprawa

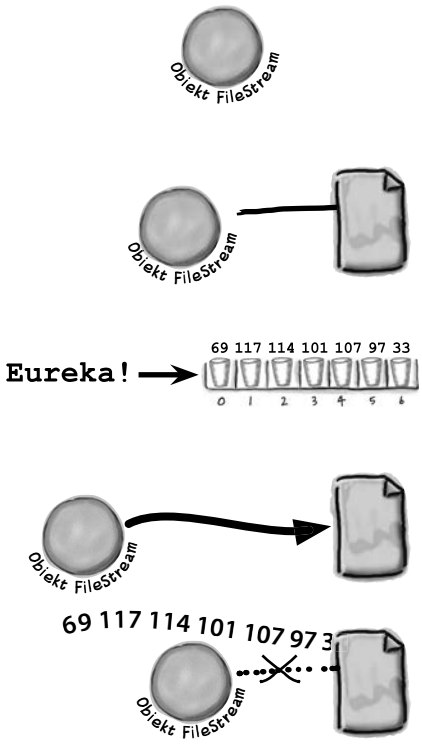
Twoim zadaniem jest stworzenie gry przygodowej, w której potężny wojownik wyrusza na misję i dzielnie walczy, poziom za poziomem, ze śmiertelnie niebezpiecznymi wrogami. Stworzysz system turowy. Oznacza to, że najpierw gracz wykonuje jeden ruch, a następnie ruch wykonuje przeciwnik. Gracz może przesunąć 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.



Odczyt i zapis plików

9 Zapisz tablice bajtów, zapisz świat

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**.



Obsługa wyjątków

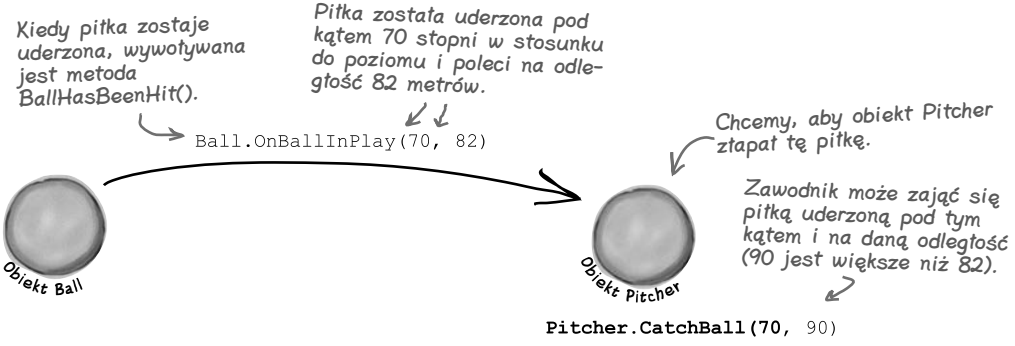
10 Gaszenie pożarów nie jest już popularne

Programiści nie mają być strażakami. Pracowałeś jak wół, przebrnąłeś przez dokumentacje techniczne 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**.

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 sprytne 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**.

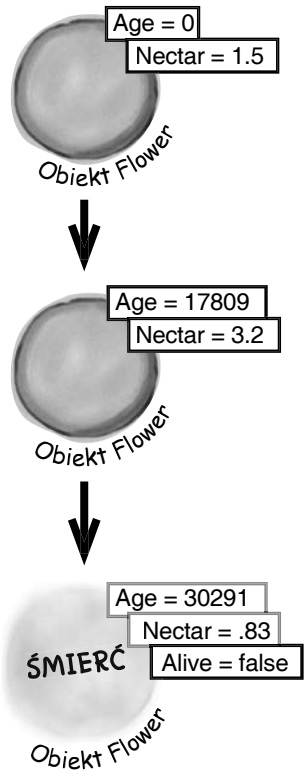


Powtórka i pokaz

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 stanowić 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**!

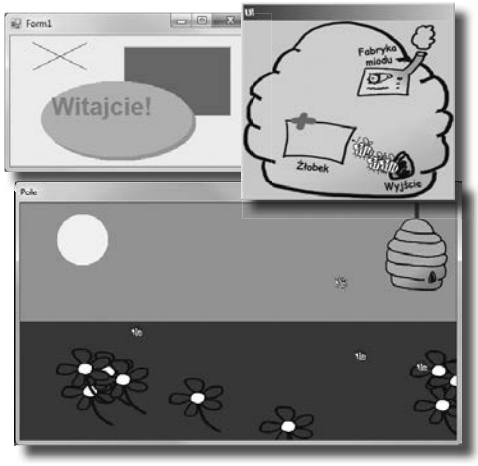
Życie i śmierć kwiatów



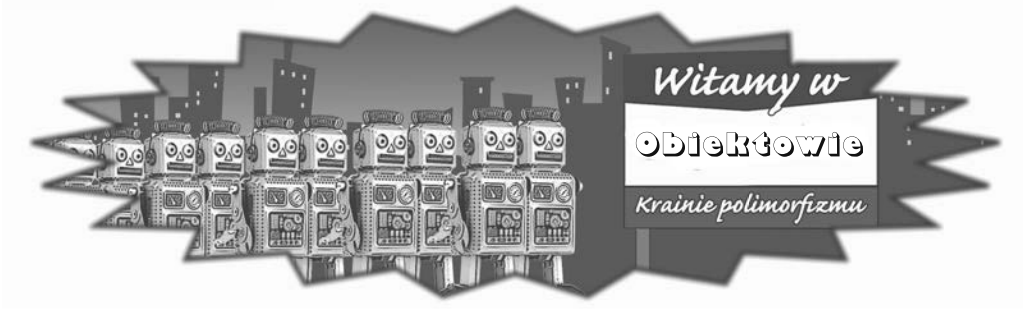
Kontrolki i grafika

13 Upiększ to

Czasami musisz wziąć sprawy grafiki we własne ręce. Przez większość czasu polegaliśmy 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.



14 Kapitan Wspaniały
Śmierć obiektu



LINQ

15 Przejmij kontrolę nad danymi

To świat przepełniony 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**.

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.

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.

S

Skorowidz

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#

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.

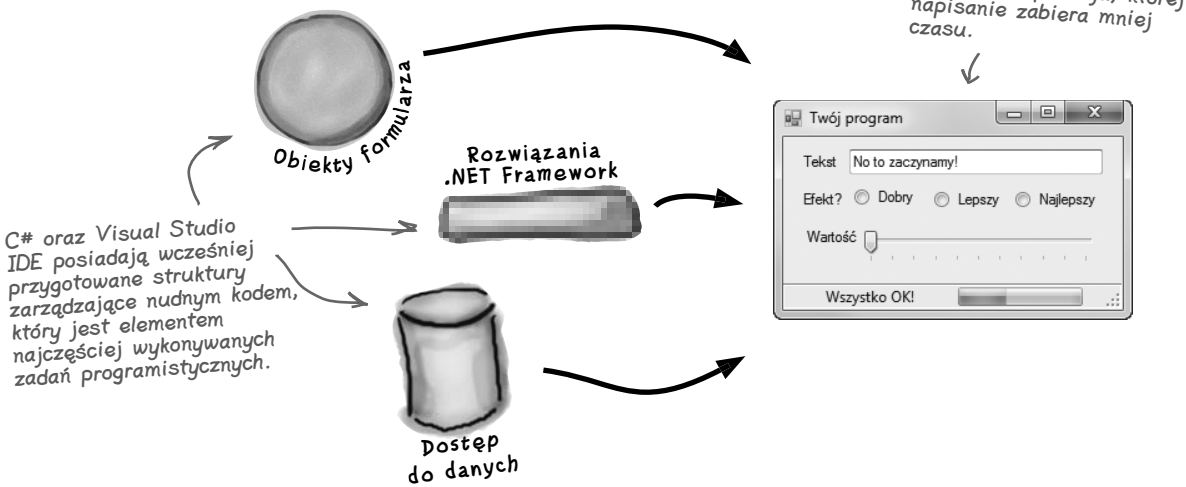
```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
namespace A_New_Program
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            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.



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:

1. **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.

2. **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.

3. **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.

4. **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:

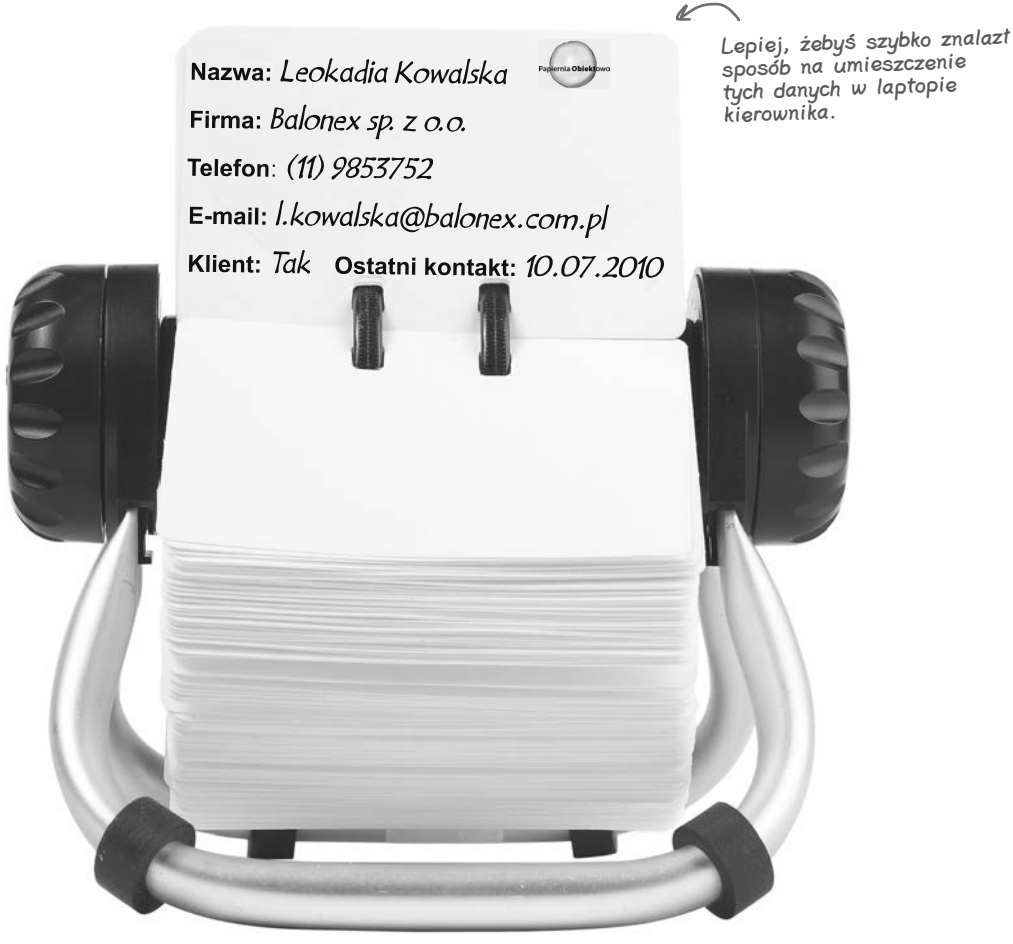
   - śledzenie wszystkich Twoich projektów,
   - ułatwianie edycji kodu,
   - kontrolowanie grafiki, dźwięków, ikon oraz innych zasobów w Twoich projektach,
   - zarządzanie i interakcja z bazami danych.

   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 znajdą 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.

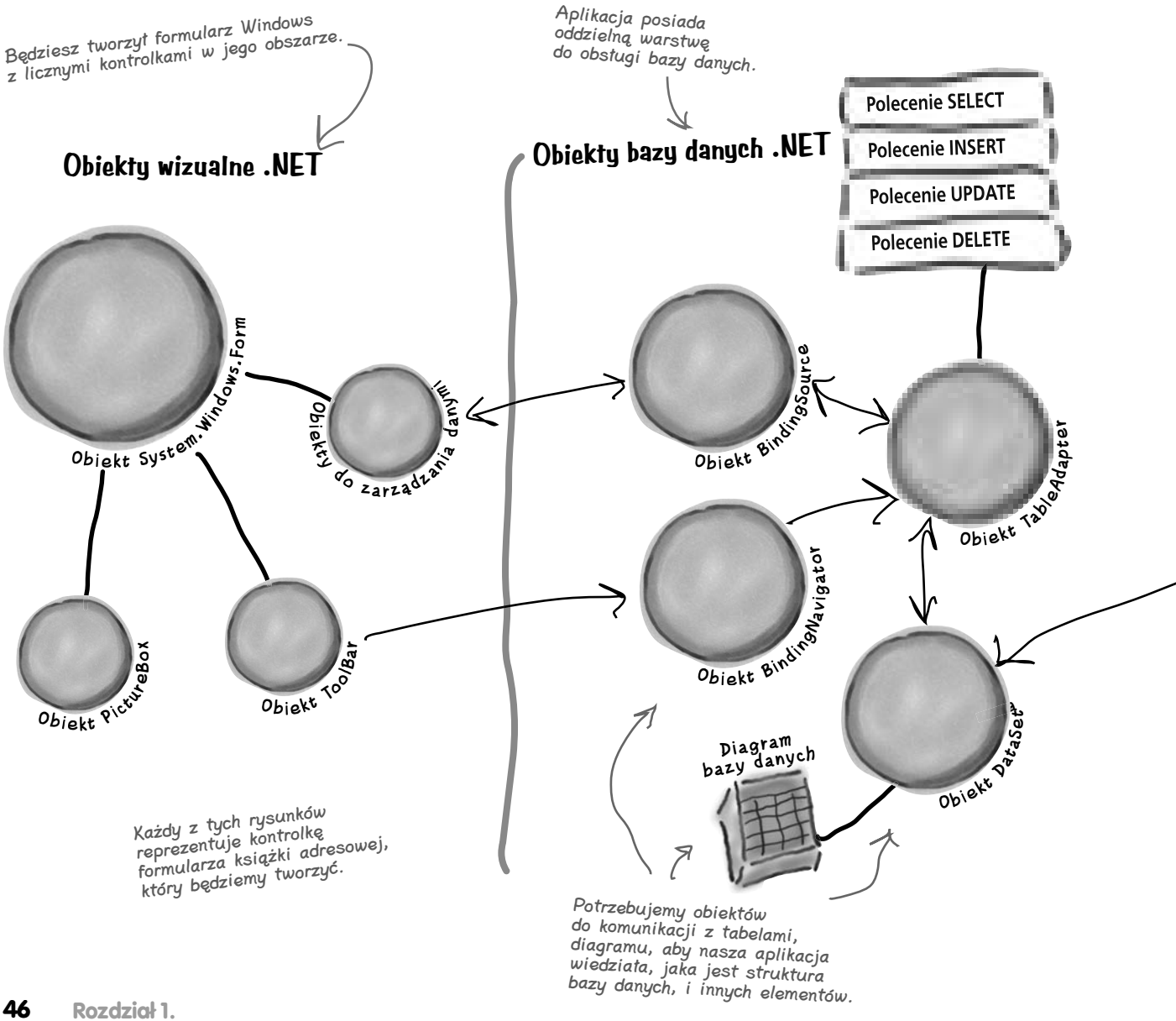
Wiemy już, że Visual Studio ułatwia pracę z bazami danych. Przechowywanie informacji w takiej formie umożliwia jednoczesny dostęp do nich dyrektorowi naczelnemu oraz działowi sprzedaży, mimo że fizycznie istnieje tylko jedna kopia danych.

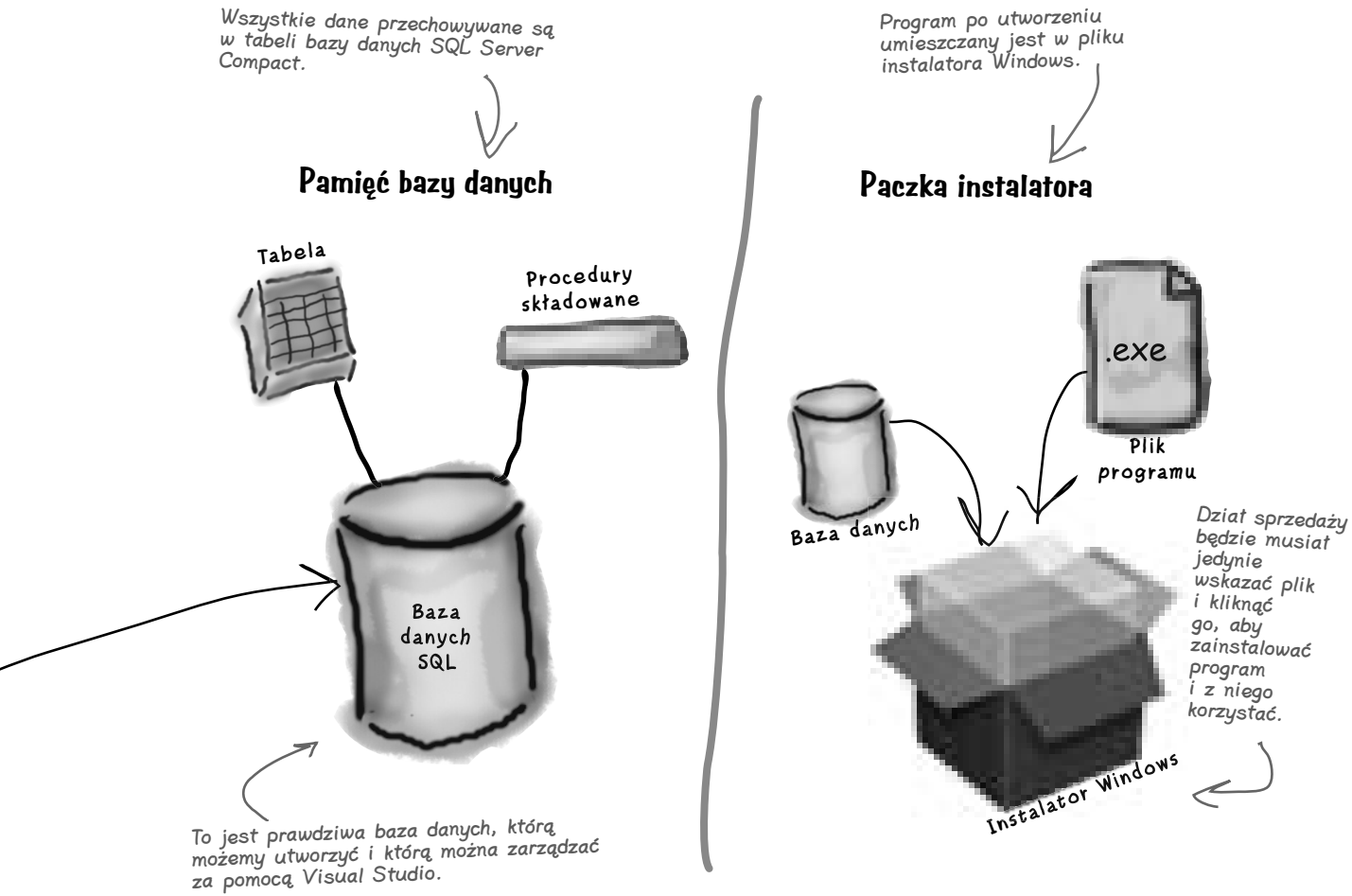
Zanim zaczniesz tworzyć kod, pomyśl o swoich użytkownikach oraz ich potrzebach; oni na pewno to docenią i będą zadowoleni z finalnego produktu, kiedy go wreszcie skończysz.

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 następnych kilku stron.

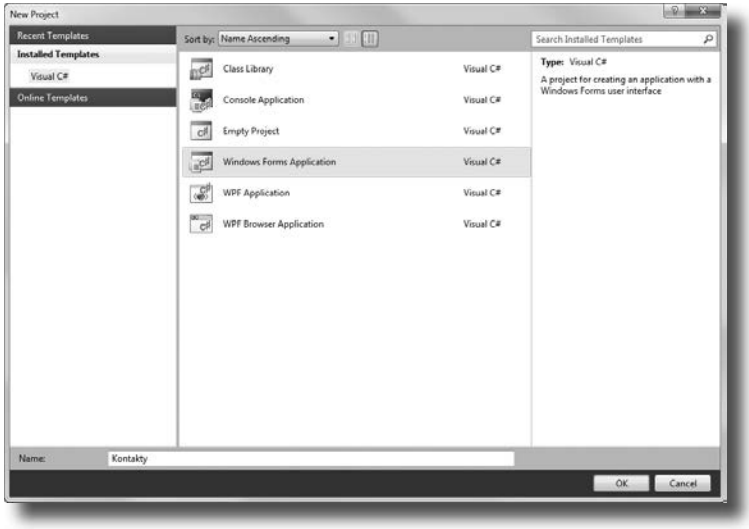
A oto struktura programu, który zamierzamy stworzyć:




Wszystkie dane przechowywane są w tabeli bazy danych SQL Server Compact.
Pamięć bazy danych
Tabela
Procedury składowane
Baza danych SQL
To jest prawdziwa baza danych, którą możemy utworzyć i którą można zarządzać za pomocą Visual Studio.
Program po utworzeniu umieszczany jest w pliku instalatora Windows.
Paczka instalatora
.exe
Plik programu
Baza danych
Instalator Windows
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”.



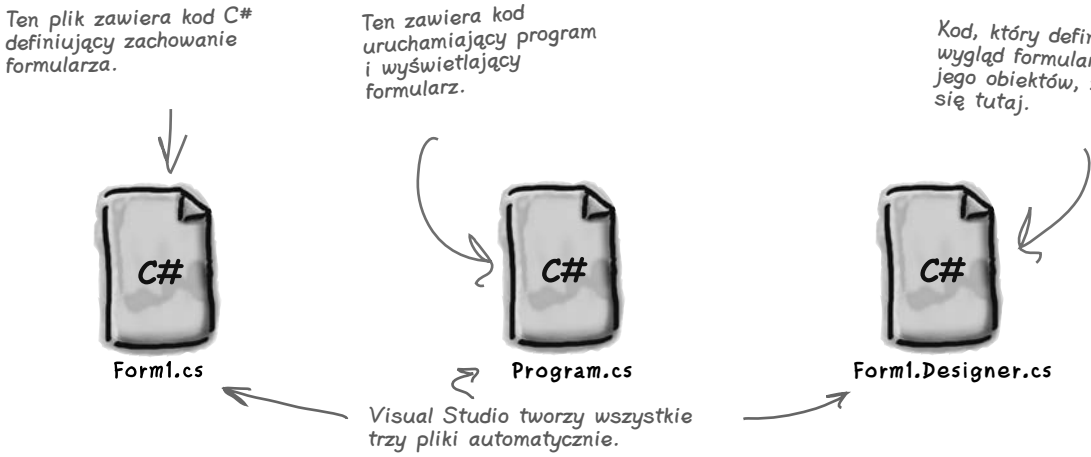
Wygląd IDE może się nieco różnić od Twojego.

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 zapisałeś 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.



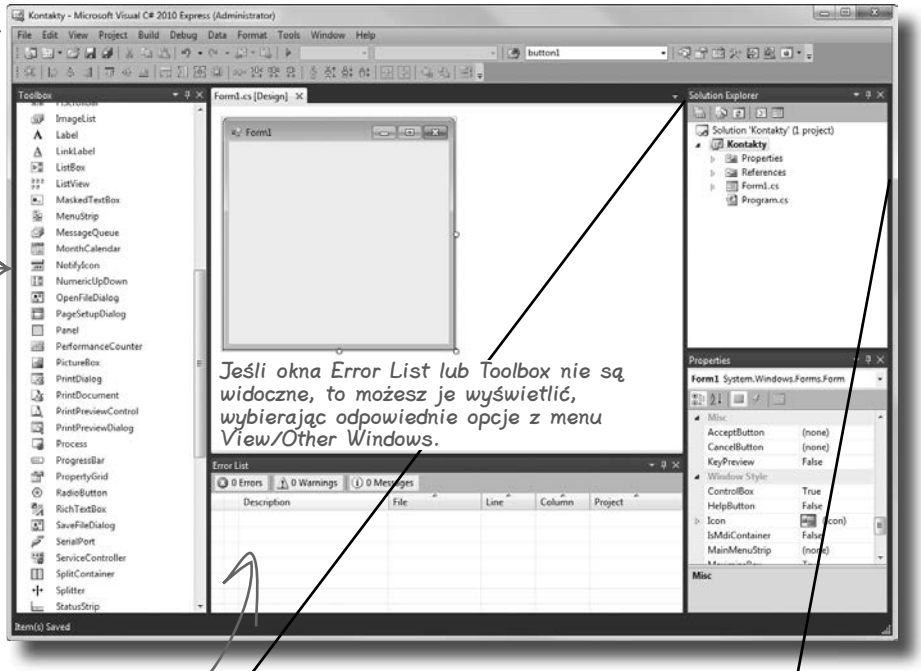
Zaostrz ołówek

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.

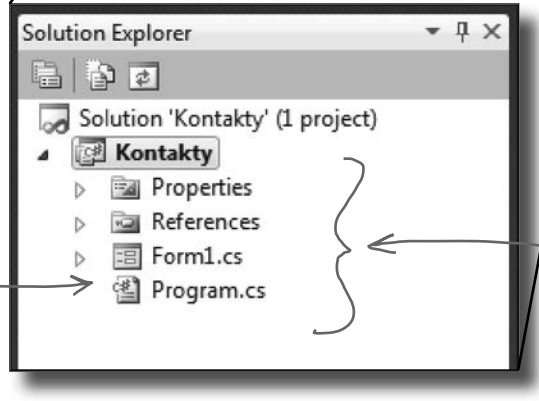
Ten pasek narzędzi posiada przyciski związane z tym, co jest aktualnie wykonywane za pomocą IDE.

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.

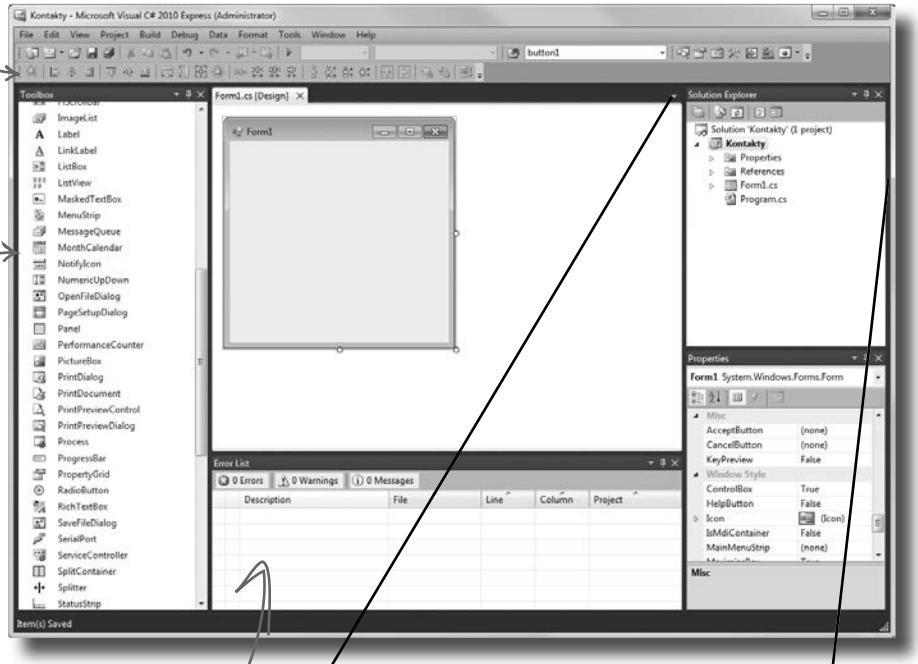


Zaostrz ołówek

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.

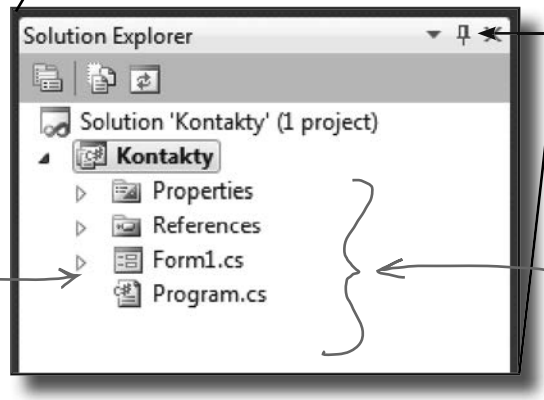
Ten pasek narzędzi posiada przyciski związane z tym, co jest aktualnie wykonywane za pomocą IDE.



To jest okno z kontrolkami. Zawiera zestaw kontrolek, 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.



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

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

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

Nie istnieją głupie pytania

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

O: Nie. IDE jest wspaniałe 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 kontrolek na formularzu. Istnieje jednak znacznie trudniejsza część programowania, czyli troska o to, co powinien 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?

O: 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?

O: 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 pożądany. 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?

O: 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?

O: Oczywiście. Podczas tworzenia nowego projektu IDE generuje domyślny formularz o nazwie *Form1* — w jego skład wchodzą 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?

O: 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.

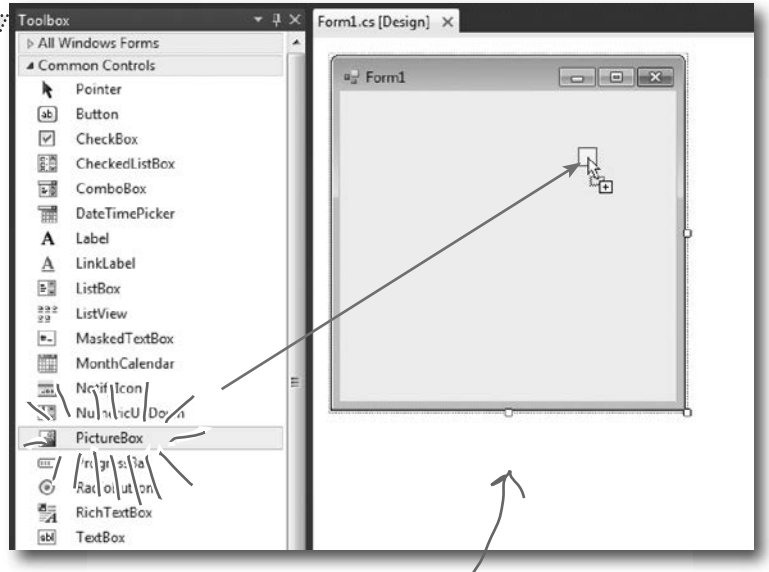
Upewnienie się, że aplikacja robi dokładnie to, do czego została stworzona, należy wyłącznie do Ciebie.

Stwórz interfejs użytkownika

Dodawanie kontrolek 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.

❶ 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.

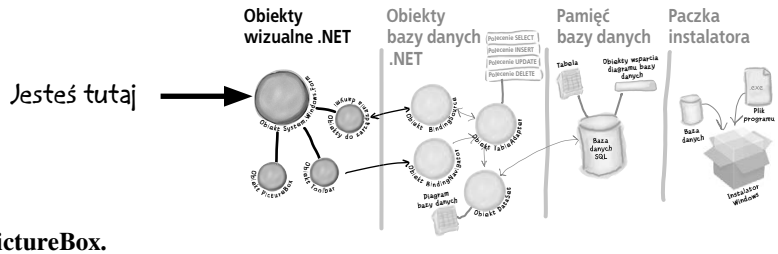


Za każdym razem, gdy wprowadzasz zmiany we właściwościach kontrolki formularza, kod w pliku Form1.Designer.cs jest zmieniany przez IDE.

Form1.Designer.cs

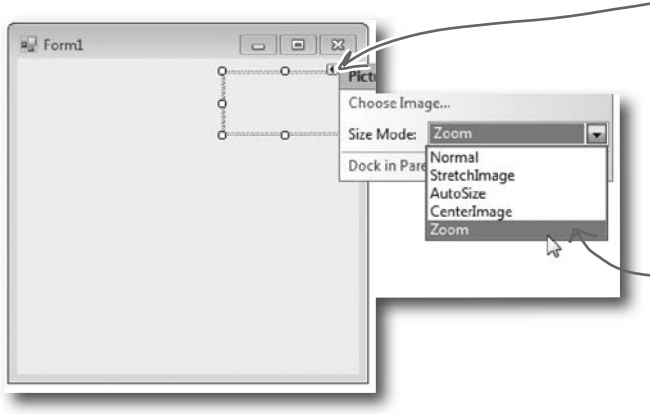
Nie przejmuj się, jeżeli nie jesteś profesjonalistą w projektowaniu graficznych interfejsów użytkownika.

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 kontrolek.



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

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

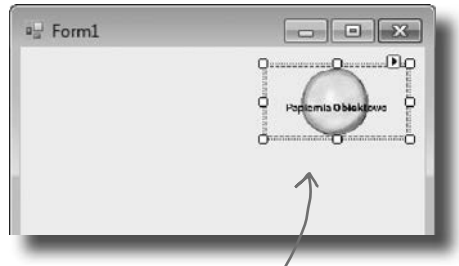
Następnie kliknij opcję Choose Image, aby wyświetlić okno dialogowe Select Resource i za jego pomocą zaimportować wybrany obrazek z lokalnego dysku.

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.

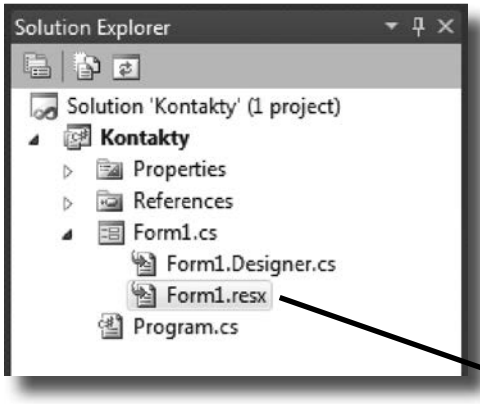
Za kulisami Visual Studio

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.

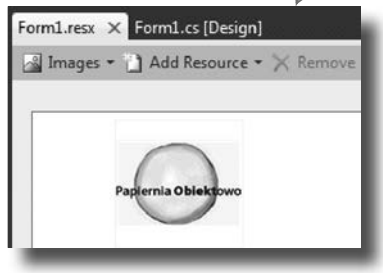


Ten obrazek jest teraz w zasobach aplikacji listy kontaktowej.

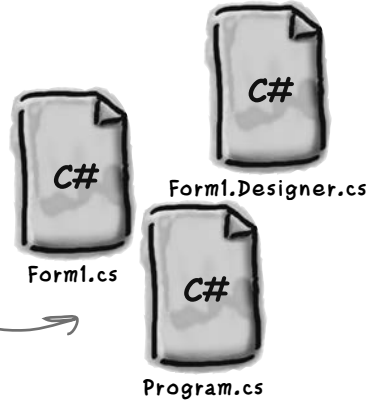
Kiedy przeciągnąłeś kontrolkę *PictureBox* na formularz, IDE automatycznie utworzyło plik *Form1.resx*, aby przechowywać zasoby i trzymać je razem w projekcie. Kliknij dwukrotnie ten plik, a zobaczysz niedawno zaimportowany obrazek.



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.

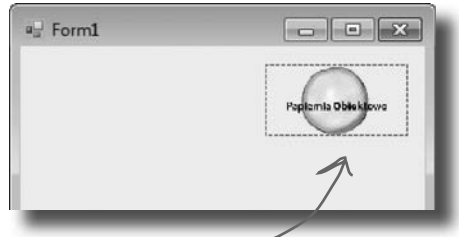


Tu są pliki, które Visual Studio wygenerowało wcześniej.

Kiedy zaimportowałeś obrazek, IDE utworzyło ten plik. Zawiera on wszystkie zasoby (grafikę, wideo, dźwięki i inne przechowywane dane) skojarzone z formularzem Form1.

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 któregokolwiek 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?

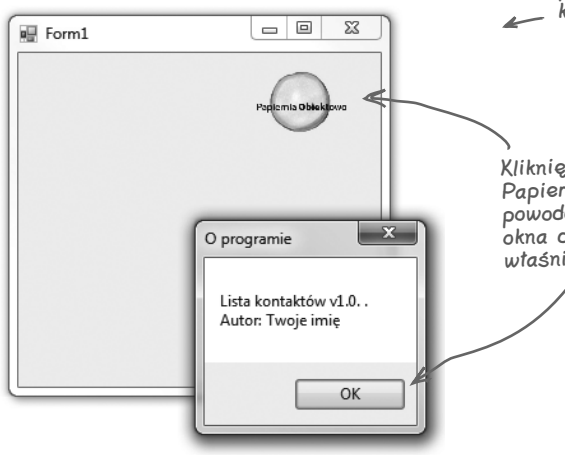
O: Metoda to po prostu *nazwany fragment kodu*. Powiemy sobie na ten temat znacznie więcej w rozdziale 2.

P: Co robi jakieś \n w kodzie?

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

Już możesz uruchomić aplikację

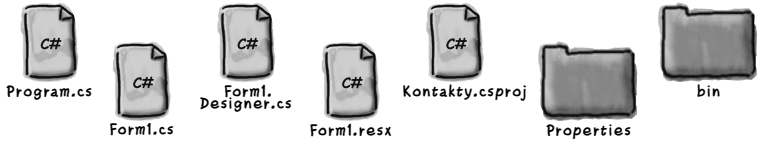
Naciśnij F5 na klawiaturze lub kliknij przycisk z zieloną strzałką ( ) na pasku narzędzi, aby sprawdzić, co udało Ci się do tego momentu napisać. (Nazywa się to „debugowaniem", czyli uruchamianiem programu za pośrednictwem IDE). Możesz zatrzymać debugowanie, wybierając *Stop Debugging* z menu *Debug*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



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 głupie 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.

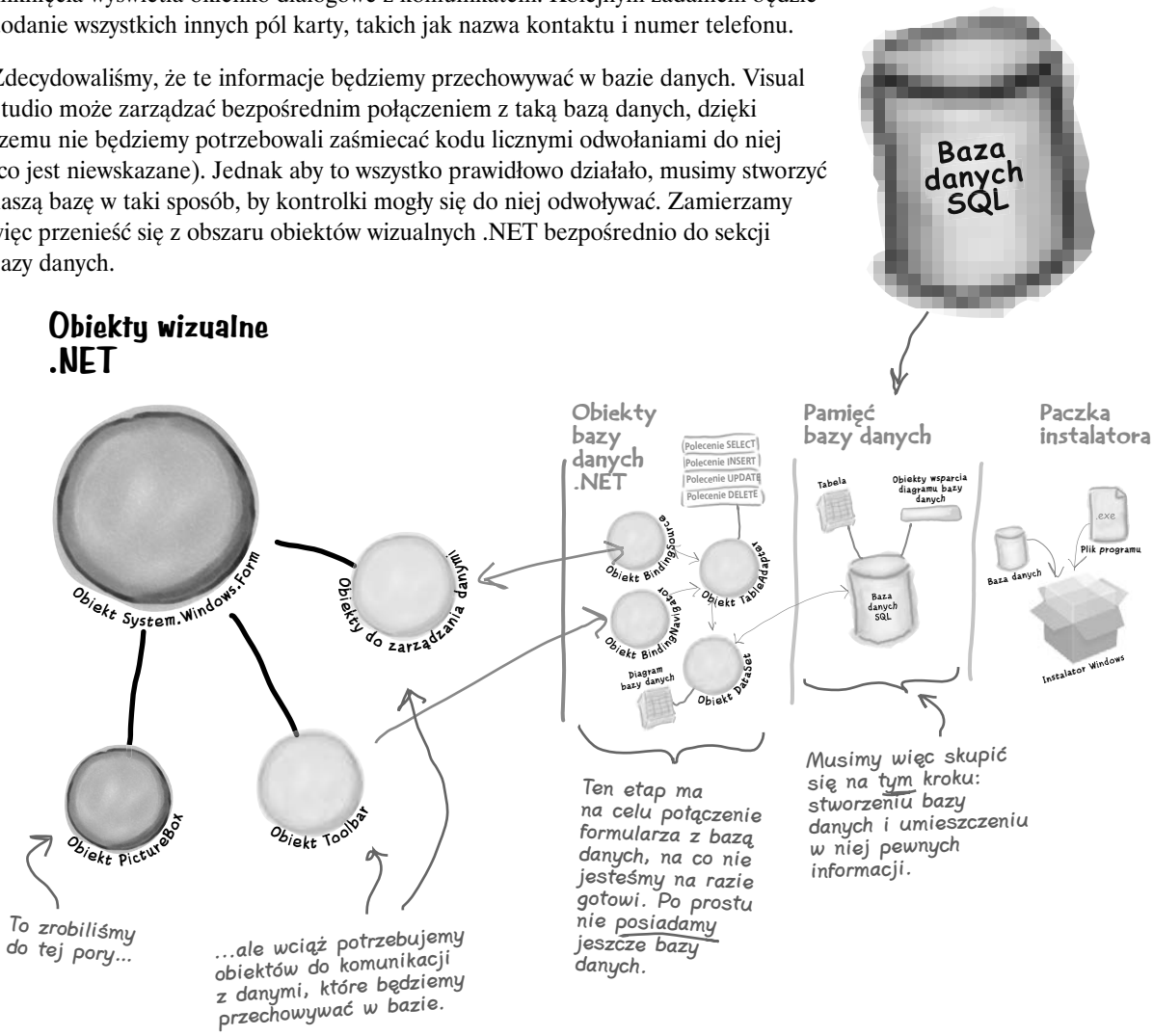
P: Nie widzę żadnego przycisku Stop Debugging na moim pasku narzędzi. O co chodzi?

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.



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

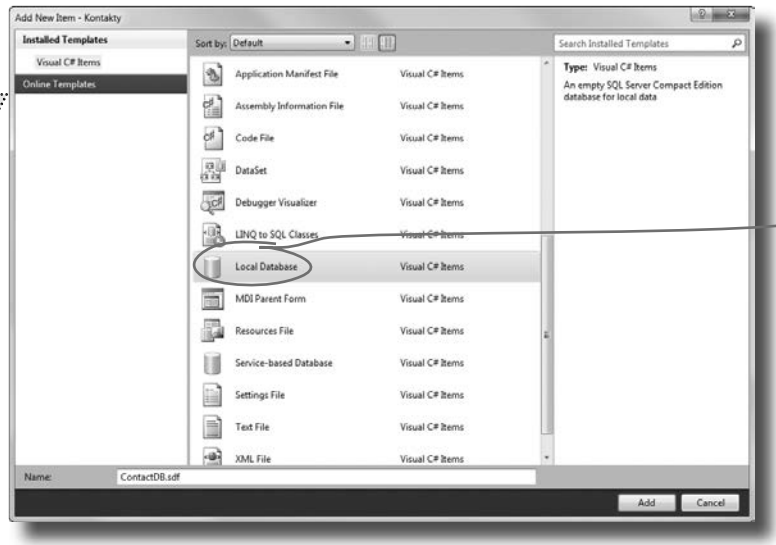
Upewnij się, że skończyłeś debugować, zanim przejdziesz dalej.

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.

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 SLQ Server Compact Edition, który będzie zawierać 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.

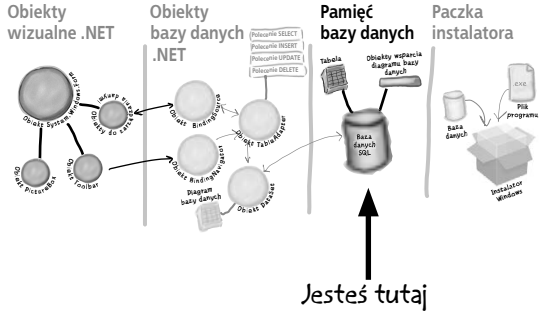
4. **Oglądnij bazę danych w oknie Solution Explorer.**
Przejdź do okna *Solution Explorer*, a zobaczysz, że baza *ContactDB* została dodana do listy plików. Kliknij dwukrotnie *ContactDB.sdf* i popatrz na lewą stronę ekranu. Toolbox został zastąpiony przez *Database Explorer*.

Uwaga!

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

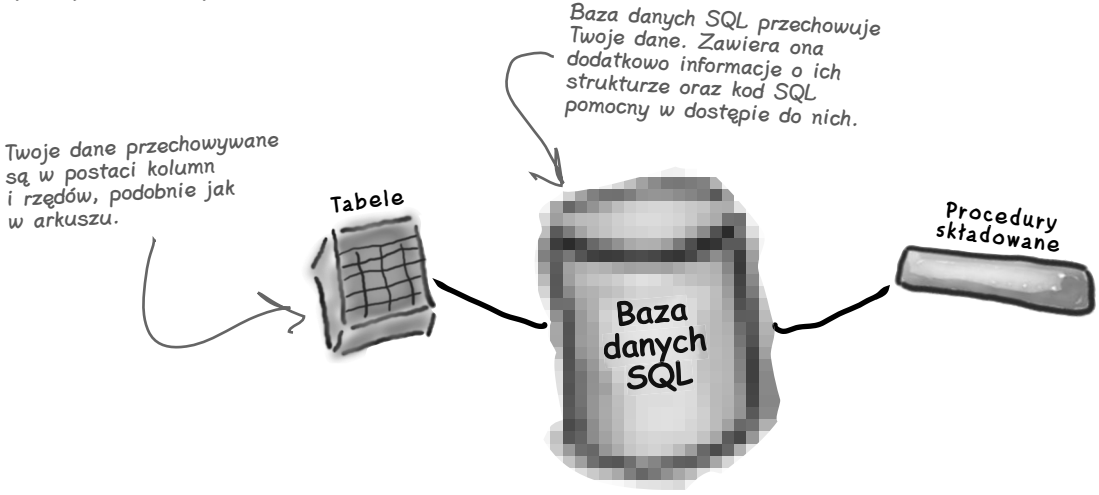
Visual Studio 2010 Professional oraz Team Foundation nie posiadają okna Database Explorer. Zamiast tego mają okno Server Explorer, które potrafi zrobić dokładnie to samo, a oprócz tego umożliwia przeglądanie danych w sieci.

IDE utworzyło bazę danych



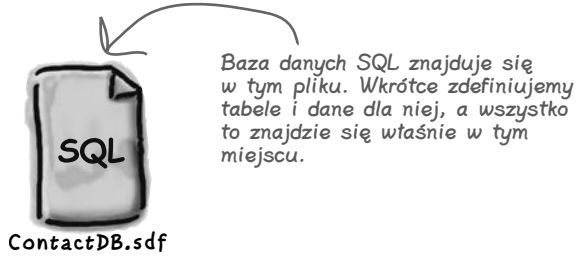
Kiedy nakazałeś IDE dodać nową bazę SQL do projektu, IDE ją utworzyło. **Baza danych SQL** to system, który przechowuje dane w sposób zorganizowany i relacyjny. Dodatkowo IDE udostępnia wszystkie narzędzia niezbędne do pracy z samą bazą danych oraz przechowywanymi w niej informacjami.

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.

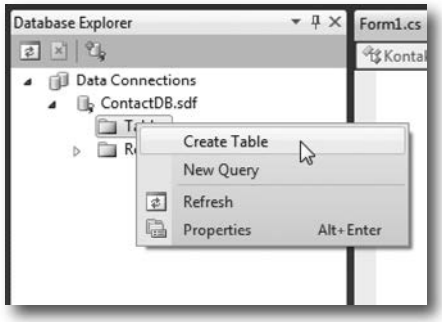


[Notka działu marketingu: Czy możemy wstawić tutaj reklamę książki „SQL. Rusz głową!”?]

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 utwórzmy 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?

O: 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 nierozerwalnie związany typ danych taki jak *String*, *Date* czy też *Bool*.

P: Do czego jest nam potrzebna kolumna ContactID?

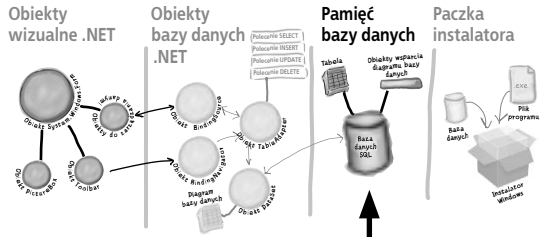
O: Umożliwia ona posiadanie unikalnego identyfikatora przez każdy rekord w większości tabel. Dlatego w przypadku przechowywania listy kontaktów do poszczególnych osób zdecydowaliśmy się utworzyć specjalną kolumnę i nazwać ją ContactID.

P: Co oznacza Int w polu Data Type?

O: 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?

O: 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!***).



Jesteś tutaj

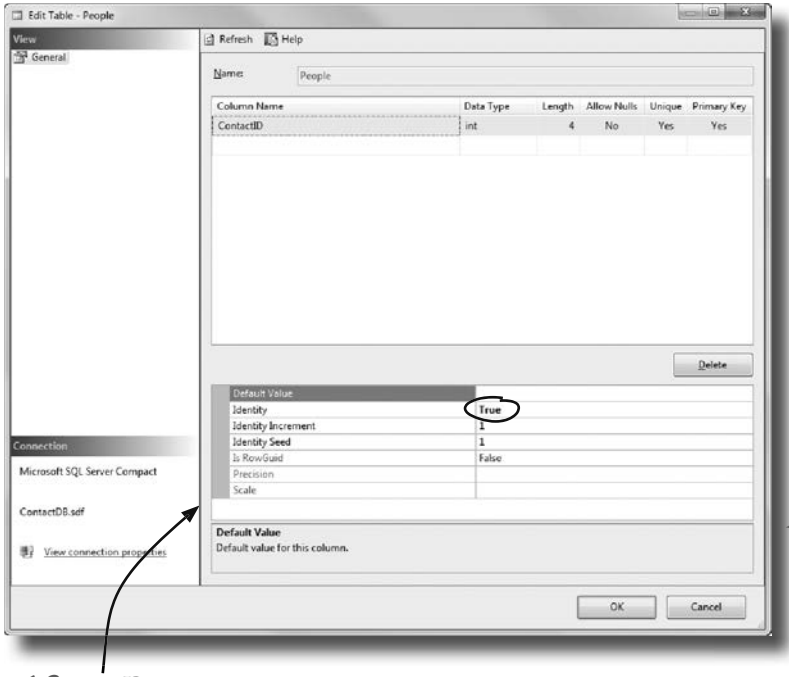
3 **Powiedz bazie danych, aby automatycznie generowała identyfikatory.**

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.



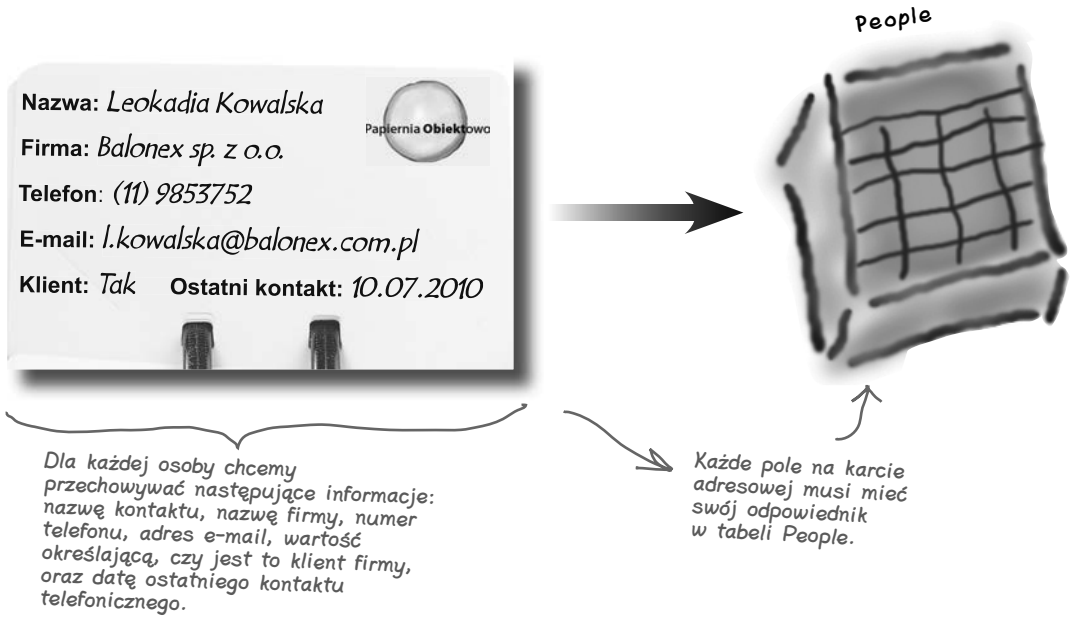
Klucz główny (ang. primary key) pomaga bazie danych szybko odnajdywać rekordy. Ponieważ jest on podstawowym narzędziem, którego program będzie używał do odnajdywania rekordów w bazie, zatem jego wartość zawsze musi być określona.

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 wyniknąć z przechowywania wielu rekordów dotyczących tej samej osoby?

KTO CO ROBI?

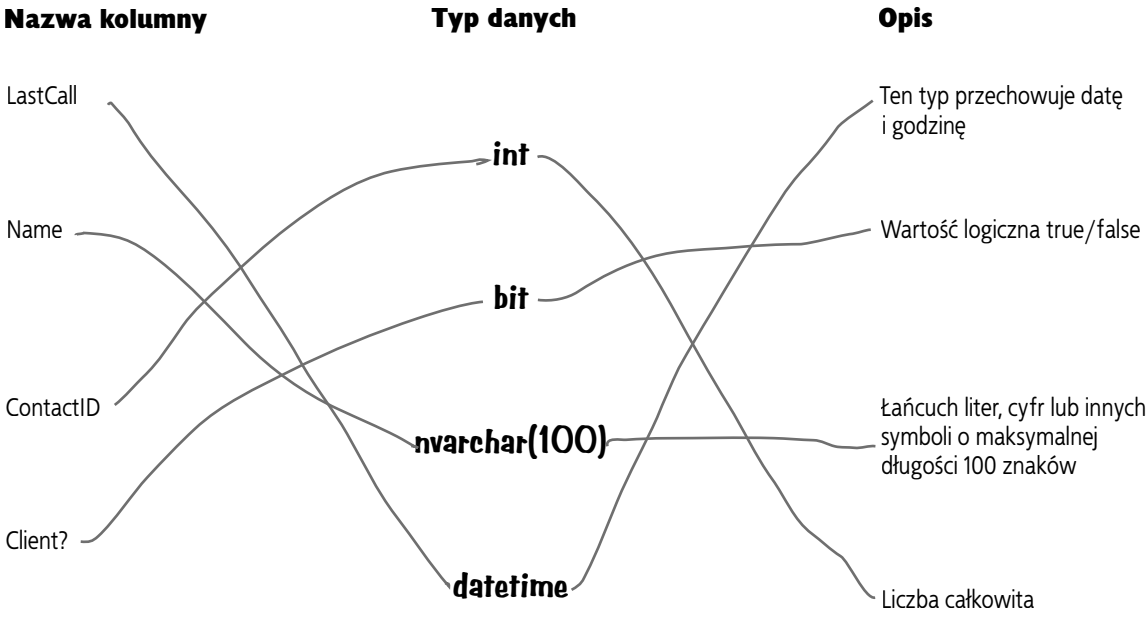
Teraz, kiedy stworzyłeś tabelę People oraz kolumnę z kluczem głównym, musisz dodać kolumny dla wszystkich pozostałych pól danych. Zobaczmy, czy potrafisz dopasować określone dane do odpowiedniej kolumny w tabeli oraz dobrać opis właściwy dla ich typu.

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

KTO CO ROBI?

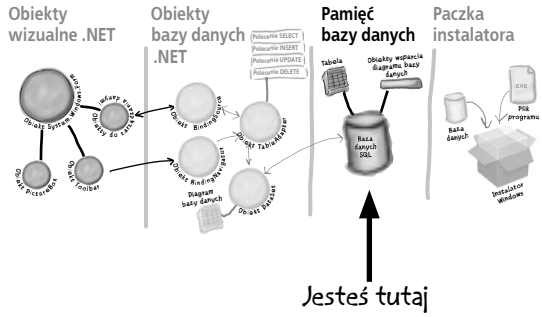
Rozwiązanie

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:



Name: People

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

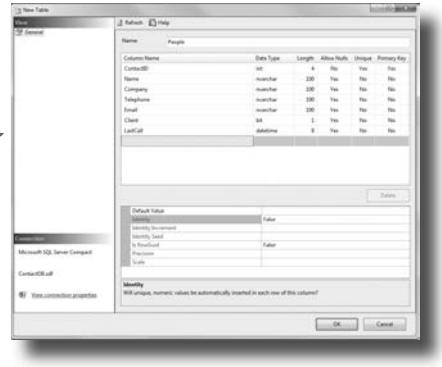
Pola bitowe przechowują wartości True lub False i mogą być reprezentowane przez kontrolkę CheckBox.

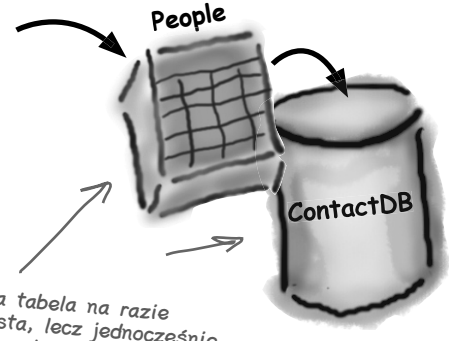
Jeżeli usuniesz zaznaczenie pola Allow Nulls, kolumna będzie musiała posiadać wartość.

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.

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

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

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

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

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

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*.

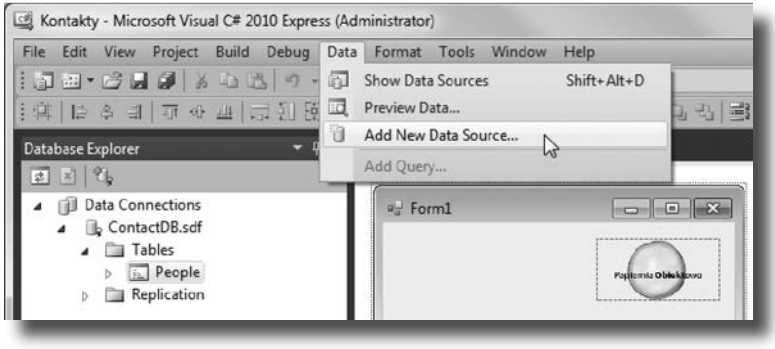
1 **Przejdź z powrotem do formularza aplikacji.**
Zamknij tabelę *People* i diagram bazy danych *ContactDB*. Powinieneś w tej chwili widzieć zakładkę *Form1.cs [Design]*.

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

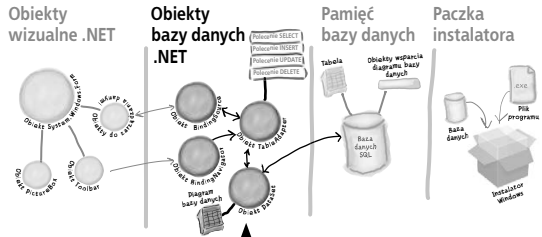
People: Query(C:\U...kty\ContactDB.sdf)

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.byrczewska@...	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	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 tworzysz, będzie obsługiwało całą komunikację pomiędzy formularzem i bazą danych.



3 **Skonfiguruj źródło danych.**

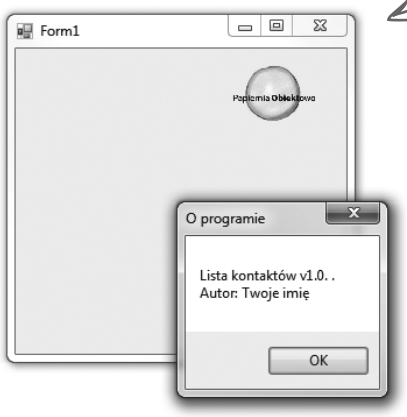
Teraz musisz dostosować źródło danych, aby używało bazy *ContactDB*. Poniżej opisano czynności, które w tym celu powinieneś wykonać:

- 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*.

Przejście 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.

ContactDBDataSet.xsd

ContactDBDataSet.Designer.cs

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

ContactDB.sdf

Ten plik to Twoja baza danych.

Dodaj kontrolki powiązane z bazą danych do formularza

Wymaga to nieco wysiłku, ale wracamy tutaj, aby utworzyć obiekty formularza, które będą współpracować z bazą danych.

Teraz możemy powrócić do naszego formularza i dodać kilka kontrolek. 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 kontrolek powiązanych z bazą danych:

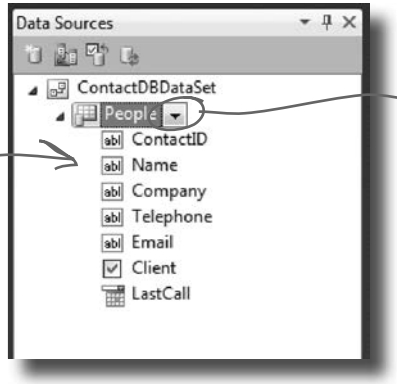
1. **Wybierz źródło danych, którego chcesz używać.**
Wybierz *Show Data Sources* z menu *Data*. Spowoduje to pojawienie się okna *Data Sources* z wyświetlonymi źródłami danych skonfigurowanymi dla tej aplikacji.

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.

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.

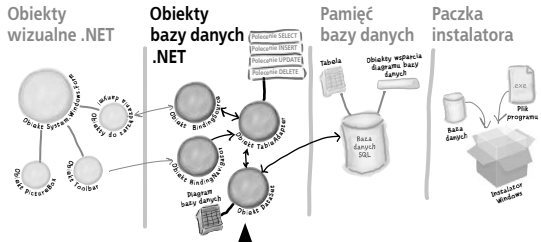
2. **Wybierz tabelę People.**
Pod elementem *ContactDBDataSet* powinieneś zobaczyć tabelę *People* i wszystkie jej kolumny. Kliknij strzałkę widoczną obok nazwy tabeli, aby wyświetlić jej zawartość — w ten sposób zobaczysz wszystkie kolumny dodane wcześniej. Kiedy w oknie *Data Sources* klikniesz tabelę *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*.



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.

Tutaj powinny zostać wyświetlone wszystkie utworzone wcześniej kolumny tabeli.

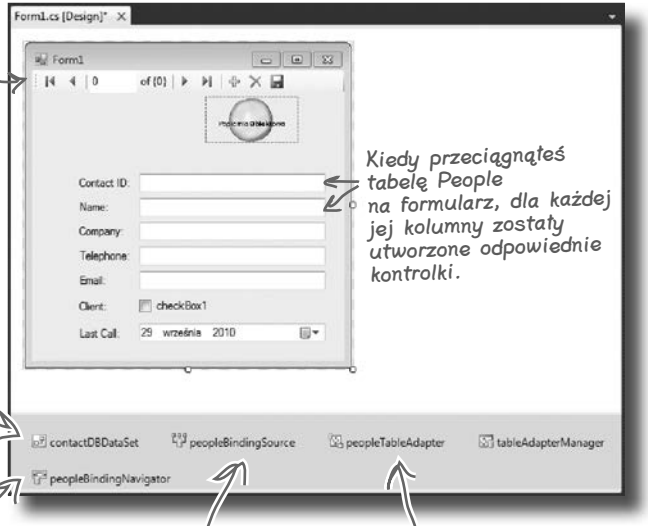
Tę rozwijalną listę zobaczysz wyłącznie w przypadku, gdy otworzone jest okno do projektowania formularzy. Pozwala ono na bezpośrednie **przeciągnięcie kontrolek danych** z okna źródła danych i umieszczenie ich na formularzu.



3 **Utwórz kontrolki powiązane z tabelą People.**

Przeciągnij 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.

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

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 tączy kontrolki paska narzędzi z tabelą bazy danych.

Ten obiekt tączy formularz z tabelą People bazy danych.

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

Dobre programy są intuicyjne w użyciu

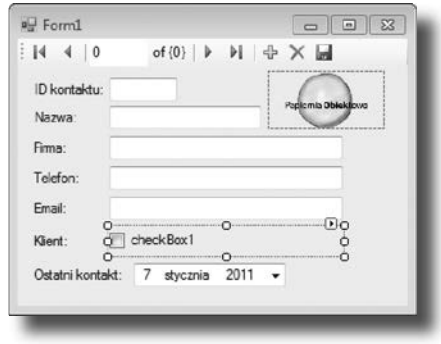
Co prawda formularz teraz działa, ale nie wygląda dobrze. Twoja aplikacja musi być więcej niż tylko funkcjonalna. Powinna być łatwa w użyciu. Dzieli Cię zaledwie kilka etapów od osiągnięcia wyglądu formularza zbliżonego do wyglądu papierowych kart, których używaliśmy na początku niniejszego rozdziału.

Nasz formularz byłby intuicyjny, gdyby wyglądał jak karta kontaktowa.

① Wyrównaj etykiety i pola tekstowe.

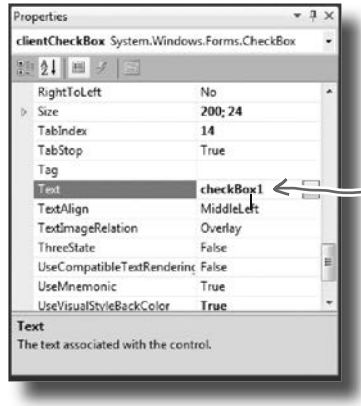
Wyrównaj etykiety i pola tekstowe względem lewej krawędzi formularza. Będzie on wyglądał podobnie do innych aplikacji, a użytkownicy będą czuli się znacznie bardziej komfortowo podczas korzystania z programu. Przetłumacz angielskie treści etykiet powstałych automatycznie na podstawie nazw kolumn z tabeli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

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

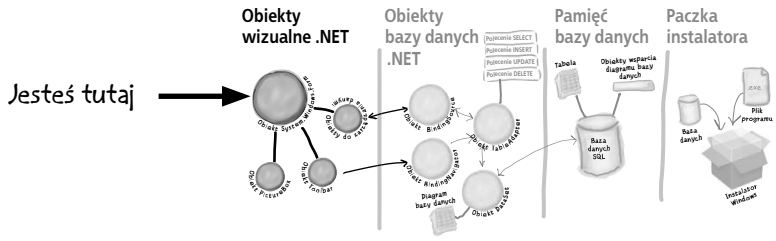


② Zmień właściwość Text kontrolki CheckBox.

Podczas wstawiania kontrolek 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.

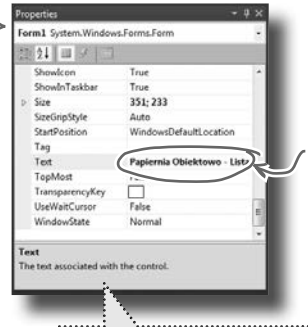


③ **Nadaj aplikacji profesjonalny wygląd.**
Możesz także zmienić nazwę formularza. Dokonasz tego poprzez kliknięcie w jego dowolnym pustym miejscu i odnalezienie właściwości *Text* w oknie *Properties*. Dla naszych potrzeb zmień nazwę na „Papiernia Obiektowo — Lista kontaktów”.

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 kontrolek 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.

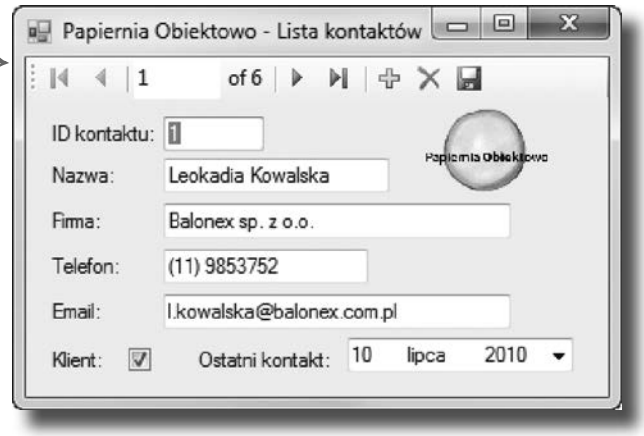
Jazda próbna

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.

Te kontrolki pozwalają przeglądać kolejne rekordy zapisane w bazie danych.



Budowanie programu nadpisuje dane w bazie.

Poświęcimy na to więcej czasu w następnym rozdziale.

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*.

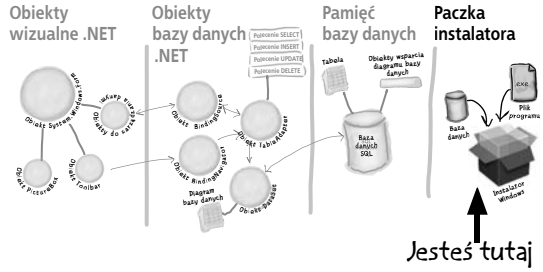
W tym przypadku plik wykonywalny oraz baza danych SQL znajdą się w katalogu *bin/debug*. Plik bazodanowy jest kopiowany za każdym razem, więc zmiany wprowadzone w trakcie pracy z IDE zostaną utracone. Jeżeli plik wykonywalny będzie uruchamiany bezpośrednio z poziomu systemu Windows, dane zostaną zachowane — 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*.

Uwaga!

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.

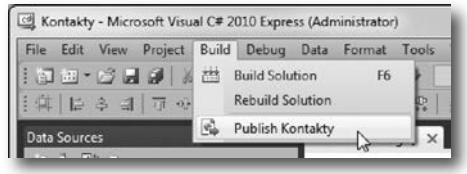
Jak zamienić TWOJĄ aplikację w aplikację WSZYSTKICH



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.

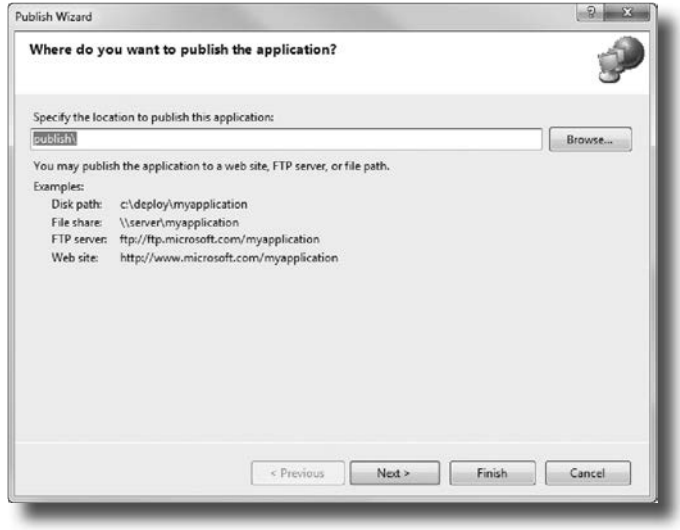
1. Wybierz *Publish Kontakty* z menu *Project*.



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.



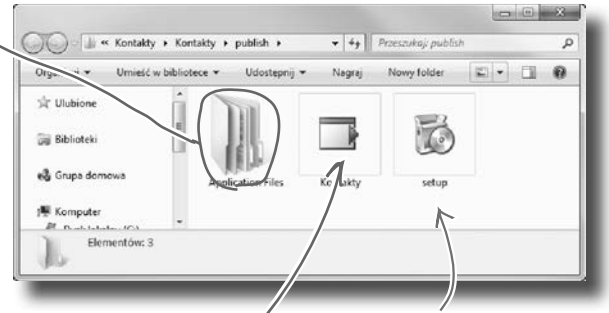
Przekaż aplikację innym użytkownikom

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.



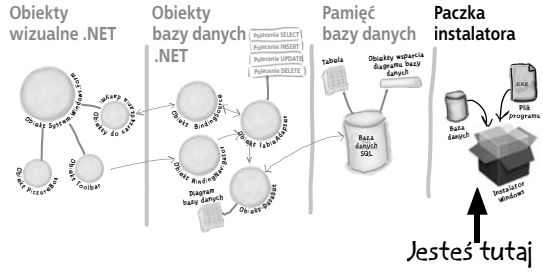
Ten plik udziela instalatorowi wszelkich informacji na temat tego, które pliki powinny zostać dołączone do programu podczas instalacji.

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

Moja sekretarka właśnie mi powiedziała, że skończyłeś 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

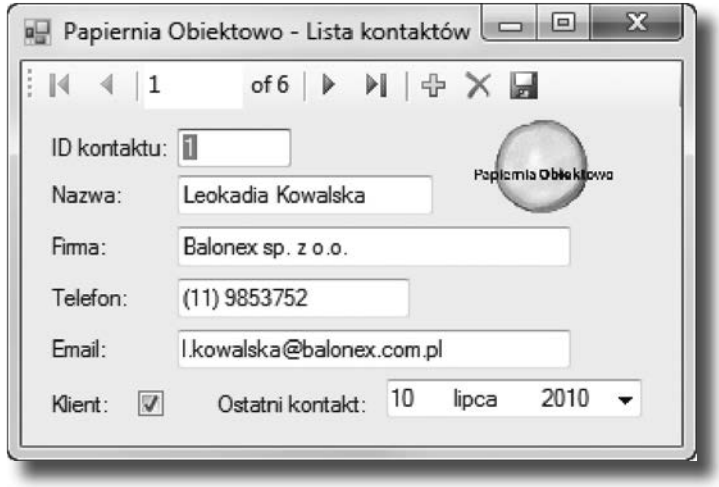


Zanim zaczniesz strzelać korkami od szampana, musisz przetestować wdrożenie i instalację. Nie dasz chyba nikomu swojego programu, dopóki sam go nie wypróbujesz?

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.

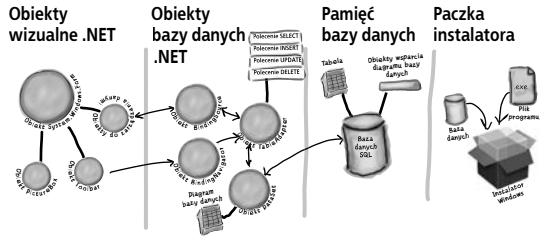
Znajdziesz tu wszystkie wpisane wcześniej kontakty. Są one zapisane w pliku bazy danych — ContactDB.sdf — który został zainstalowany wraz z Twoim programem.

TESTUJ WSZYSTKO!

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

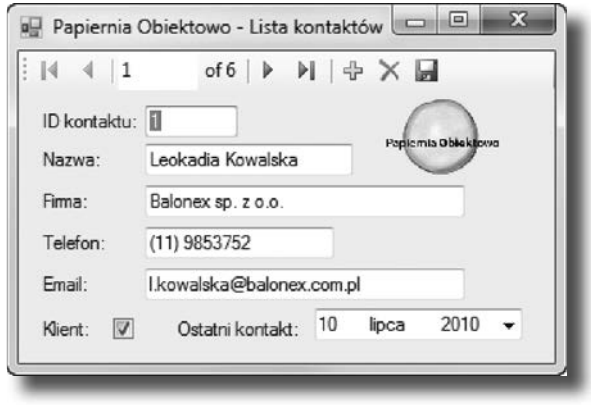
Stworzyłeś pełnowartościową aplikację bazodanową

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 licznych okien, przycisków i zestawu instrukcji SQL.

Skorowidz

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

X

Y

Z

Ź

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 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 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 debugerem
- Pętle, instrukcje warunkowe
- Elementy programowania obiektowego
- Typy zmiennych
- Referencje
- Tablice
- Hermetyzacja obiektów
- Implementacja interfejsów
- Typy wyliczeniowe
- 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

Nr katalogowy: 5873

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

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

Cena 99,00 zł

ISBN 978-83-246-2953-4

9 788324 629534

Informatyka w najlepszym wydaniu