Head First C#. Edycja polska

Autor: Andrew Stellman, Jennifer Greene
Tłumaczenie: Paweł Dyl
ISBN: 978-83-246-1546-9
Tytuł oryginału: Head First C# (Head First)
Format: 200x230, stron: 752

Język programowania C# został zaprojektowany specjalnie dla firmy Microsoft. C# czerpie najlepsze wzorce z języka Java oraz C++. Aktualnie dostępna wersja 3.0 zawiera takie elementy jak automatyczne oczyszczanie pamięci, typy ogólne, dynamiczne tworzenie kodu i wiele innych. Język C# zawiera bogatą bibliotekę klas pozwalających na tworzenie i rozwijanie aplikacji okienkowych, bazodanowych, a także dynamicznych aplikacji internetowych. Rozwiązanie to zdobyło już swoją pozycję na rynku języków programowania, a narzędzia dla programistów dostarczane przez firmę Microsoft sprawiają, że pozycja ta wydaje się niezagrożona.

W książce „Head First C#. Edycja polska" autorzy, jak zwykle w charakterystyczny dla tej serii – niekonwencjonalny, a przy tym niezwykle skuteczny sposób, nauczą Cię niezbędnych podstaw C#. Lektura tej książki pozwoli Ci na swobodne poruszanie się wśród takich zagadnień jak wykorzystanie interfejsów czy też dziedziczenie. Dowiesz się, w jaki sposób obsługiwać wyjątki oraz zapisywać dane do pliku, aby potem je z niego odczytać. Nauczysz się korzystać z języka LINQ, a także odbędziesz krótki kurs korzystania z Visual Studio. Sposób, w jaki została napisana ta książka, gwarantuje, że szybko i z łatwością opanujesz język C#!

- Podstawy C#
- Typy danych wykorzystywane w C#
- Wykorzystanie interfejsów oraz klas abstrakcyjnych
- Zastosowanie typów wyliczeniowych i kolekcji
- Sposób wykorzystania plików do przechowywania danych
- Wykorzystanie zdarzeń i delegacji
- Zastosowania języka LINQ
- Visual Studio – sposób na wydajniejsze tworzenie aplikacji

Poznaj język C# szybko, łatwo i przyjemnie!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści (streszczenie)

Spis treści (właściwy)

Wstęp

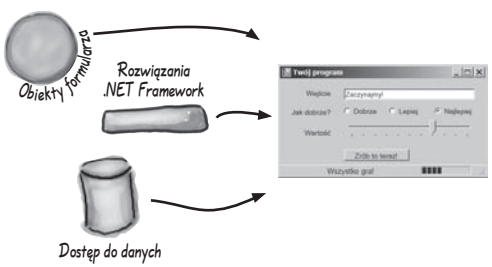
Przygotuj się na C#. Właśnie sobie siedzisz i próbujesz się czegoś nauczyć, ale mózg wciąż powtarza Ci, że cała ta nauka nie jest nic warta. Twój umysł mówi: „Lepiej wyjdź z pokoju i zajmij się ważniejszymi sprawami, takimi jak to, których dzikich zwierząt unikać, oraz to, że strzelanie z łuku na golasa 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, pozwalający przyciskowi po raz kolejny zadziałać. I to nie wszystko. Dodatkowo będziesz mógł skupić się na **faktycznym wykonywaniu swojej pracy,** zamiast zajmować umysł, pamiętając, który parametr metody odpowiadał za *nazwę* przycisku, a który był odpowiedzialny za *wyświetlany na nim tekst*. Brzmi zachęcająco? Przewróć zatem stronę i przystąpmy do programowania.

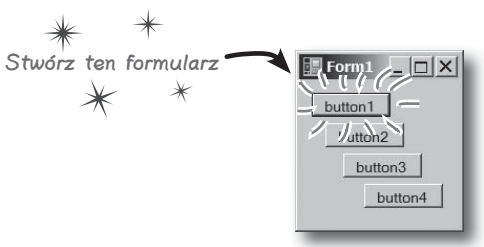


To tylko kod

2 Pod maską

Jesteś programistą, nie tylko użytkownikiem IDE.

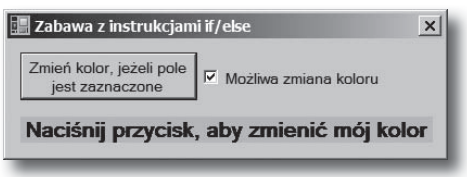
IDE może wykonać za Ciebie wiele pracy, ale na razie jest to wszystko, co może dla Ciebie zrobić. Oczywiście, istnieje wiele **powtarzalnych czynności** podczas pisania aplikacji i IDE okazuje się tu bardzo pomocne. Praca z nim to jednak *dopiero początek*. Możesz wycisnąć ze swoich programów znacznie 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.

Klasa zawiera **kawałek** kodu Twojego programu (chociaż istnieją także bardzo małe programy, które składają 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.

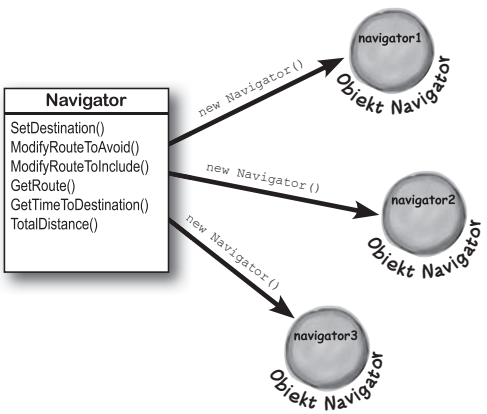


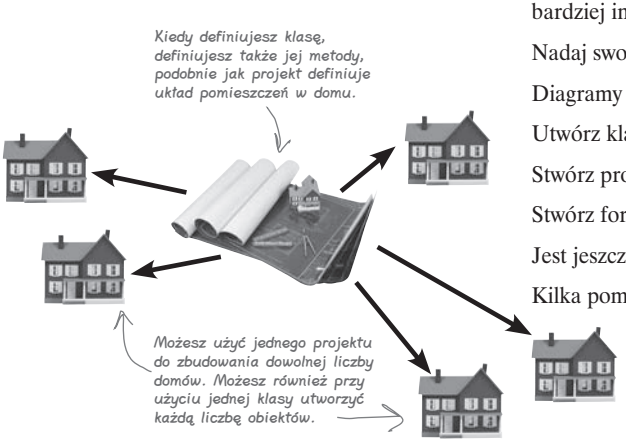
Obiekty: zorientuj się!

3 Tworzenie kodu ma sens

Każdy pisany przez Ciebie program rozwiązuje jakiś problem.

Podczas pisania programu staraj się zawsze zastanowić nad tym, jaki *problem* powinien on rozwiązywać. Właśnie do tego przydają się ***obiekty***. Pozwalają one na odpowiedni podział struktury kodu, aby same mogły się zająć rozwiązywaniem problemów. Twój czas jest przeznaczany na wykonywanie *rzeczywistej pracy* i nie jest przerywany przedzieraniem się przez morze kodu. Prawidłowe użycie obiektów spowoduje, że proces pisania kodu stanie się bardziej *intuicyjny*, natomiast późniejsze czytanie i zmiany będą łatwiejsze.

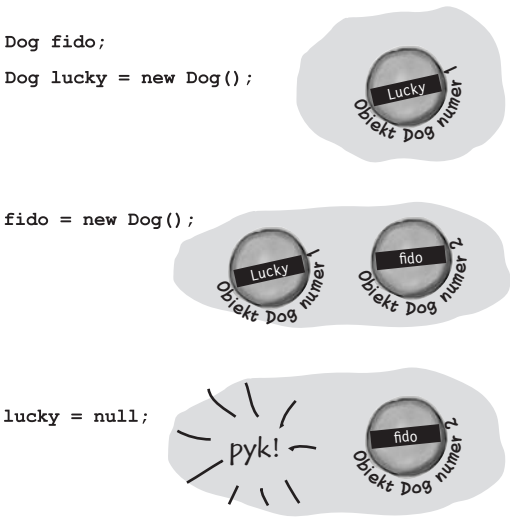




Typy i referencje

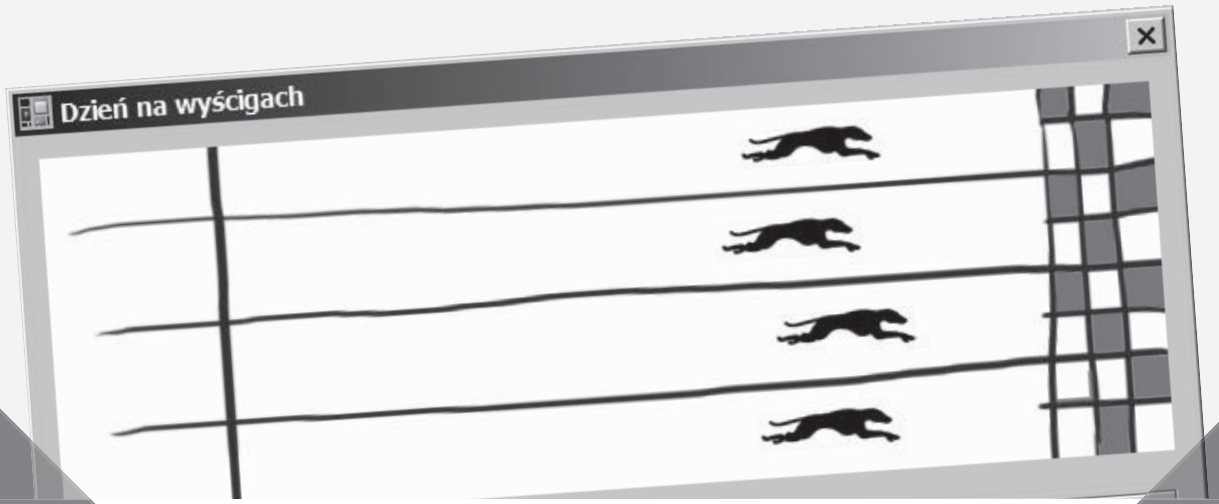
4 Jest 10:00. Czy wiesz, gdzie są Twoje dane?

Bez danych Twoje programy są bezużyteczne. Potrzebujesz od użytkowników informacji. 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*).



Laboratorium C# nr 1

Joe, Bob i Al uwielbiają chodzić na tor wyścigowy, ale ciągła utrata pieniędzy powoduje u nich frustrację. Potrzebują symulatora, aby mogli określić zwycięzcę, zanim pozbędą się wszystkich pieniędzy. Jeśli dobrze wywiążesz się z zadania, zostaniesz wprowadzony w arkana ich sposobu zarabiania.

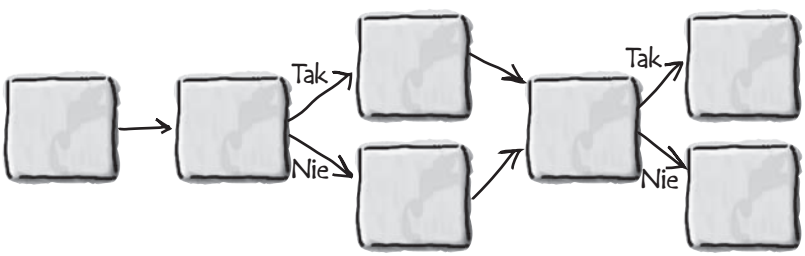


Hermetyzacja

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**, natomiast metody, które dodasz, pozwolą Ci na **zabezpieczenie dostępu do danych**.



Dziedziczenie

6 Drzewo genealogiczne Twoich obiektów

Czasami *CHCIAŁBYŚ* być dokładnie taki sam jak Twoi rodzice.

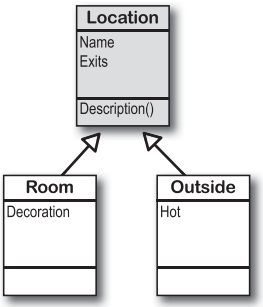
Czy kiedykolwiek natknąłeś się na obiekt, który robiłby ***prawie*** wszystko, czego byś sobie od niego życzył? Czy kiedykolwiek znalazłeś się w takiej sytuacji, że gdybyś ***zmienił dosłownie kilka rzeczy***, obiekt byłby perfekcyjny? To jest jeden z powodów, dla których powstało **dziedziczenie**, czyli jedna z najpotężniejszych koncepcji i technik w języku C#. Zanim przystąpisz do zgłębiania tajników tego rozdziału, dowiesz się, jak **rozszerzyć** obiekt, aby uzyskać dane jego zachowanie, ale równocześnie zachować jego **elastyczność** podczas zmieniania tego zachowania. Unikniesz **wielokrotnego pisania kodu, przedstawisz prawdziwy świat** znacznie dokładniej, a w wyniku otrzymasz kod **łatwiejszy do zarządzania**.

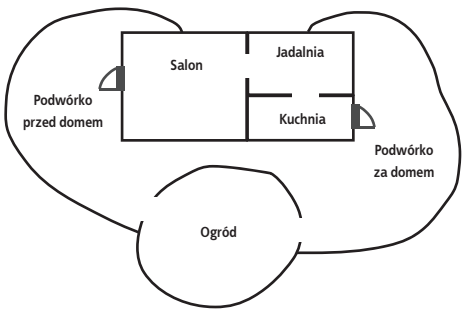
Interfejsy i klasy abstrakcyjne

7 Klasy, które dotrzymują swoich obietnic

Czyny potrafią powiedzieć więcej niż słowa.

Czasami potrzebujesz pogrupować swoje obiekty na podstawie tego, **co robią**, zamiast tego, z jakiej klasy 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 swoje zadania**... albo kompilator połamie Twoje kolana, zrozumiałeś?



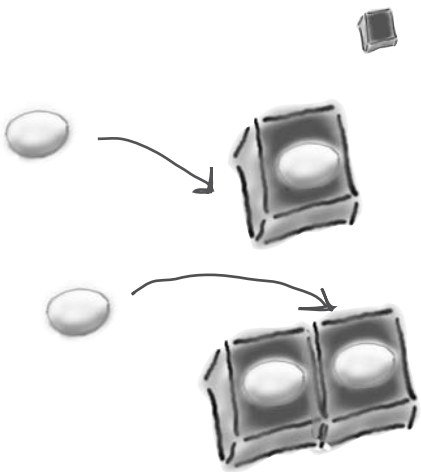


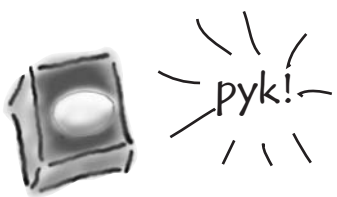
Typy wyliczeniowe i kolekcje

8 Przechowywanie dużej ilości danych

Z deszczu pod rynnę.

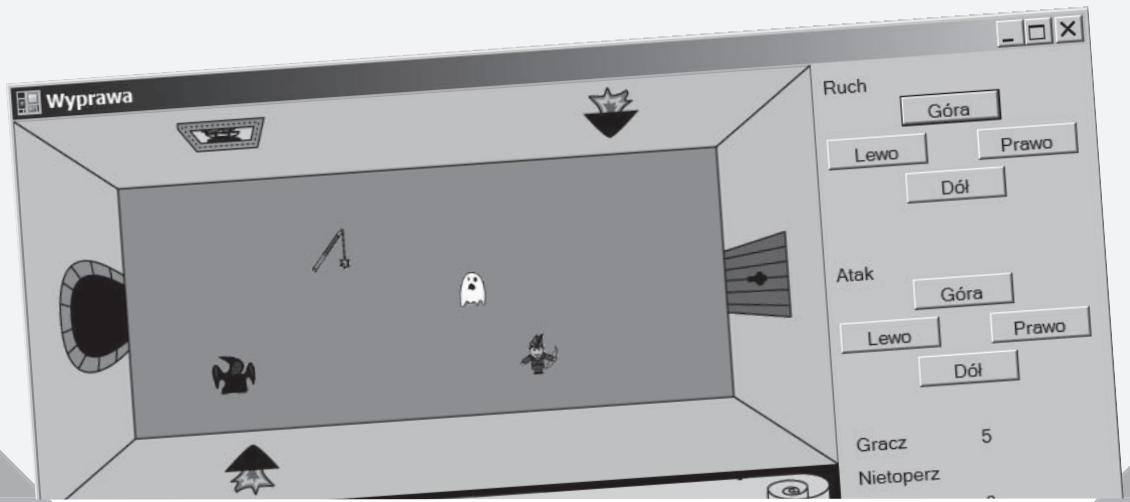
W rzeczywistym świecie nie musisz się zwykle zajmować danymi w małych ilościach i w niewielkich fragmentach. Nie, Twoje dane przychodzą do Ciebie w **grupach, stosach, pękach, kopach**. Potrzebujesz jakiegoś potężnego narzędzia do zorganizowania ich. Nadszedł czas, aby przedstawić **kolekcje**. Pozwalają one **przechowywać, sortować i 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

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 lub ataku dostaje każdy z wrogów. Gra toczy się do czasu, aż gracz pokona wszystkich przeciwników na wszystkich siedmiu poziomach lub zginie.

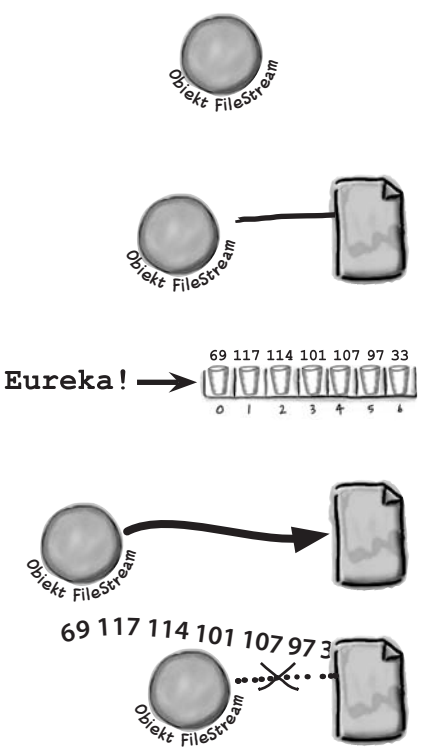


Odczyt i zapis plików

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.

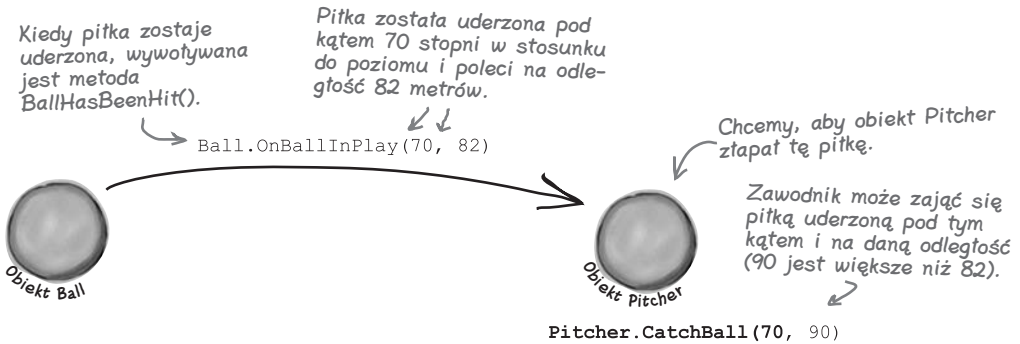
Pracowałeś jak wół, przebrnąłeś przez dokumentacje techniczne i kilka ujmujących książek *Head First*, wspiąłeś się na szczyt swoich możliwości: jesteś **mistrzem programistów**. W dalszym ciągu musisz jednak odrywać się od pracy, ponieważ **program wyłącza się** lub **nie zachowuje się tak jak powinien**. 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 masz zbyt wielu obiektów podpiętych pod to samo zdarzenie. Wtedy bardzo pomocne okazują się **funkcje zwrotne**.



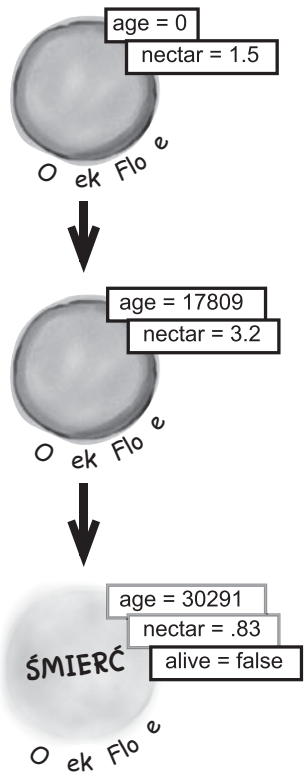
Powtórka i pokaz

12 Wiedza, moc i tworzenie ciekawych rzeczy

Uczenie się nie jest dobre, dopóki czegoś nie ZBUDUJESZ.

Dopóki nie napiszesz kodu, który działa, bardzo trudno jest *pojąć* bardziej skomplikowane techniki C#. W tym rozdziale zamierzamy przedstawić parę różnych, nowych zagadnień: **zegary** oraz zarządzanie kolekcjami przy użyciu **LINQ** (w celu zmiany nazwy). Chcemy także napisać pierwszą część **naprawdę złożonej aplikacji**, aby się upewnić, że naprawdę dobrze zrozumiałeś zagadnienia przedstawione w poprzednich rozdziałach. Zapnij pasy... czas stworzyć naprawdę **fajny program**.

Życie i śmierć kwiatów

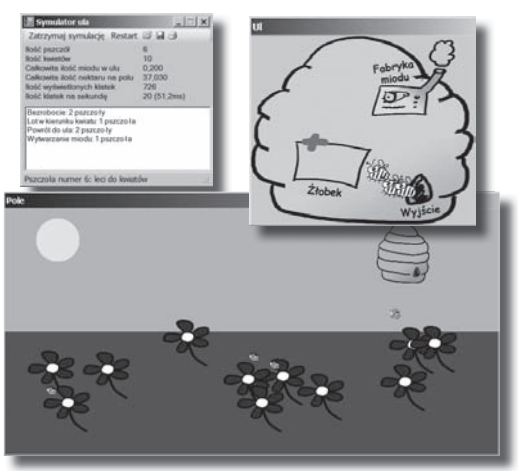


Kontrolki i grafika

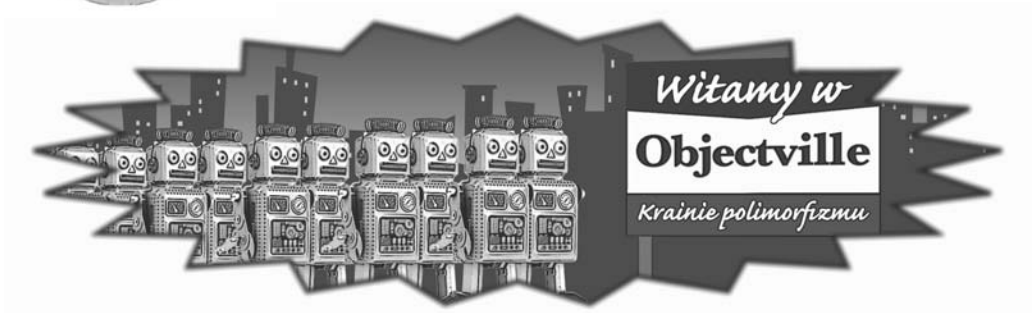
13 Upiększ to

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.

To już nie wróci. Czasy, gdy mogłeś programować kilka dni, a nawet kilka tygodni, bez konieczności pracy **z ogromem danych**. Nadeszła epoka, w której ***wszystko opiera się na danych***. W rzeczywistości dość często będziesz musiał pracować z takimi, 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 dane** i **łączyć te pochodzące z różnych źródeł**.

Laboratorium C# numer 3

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.

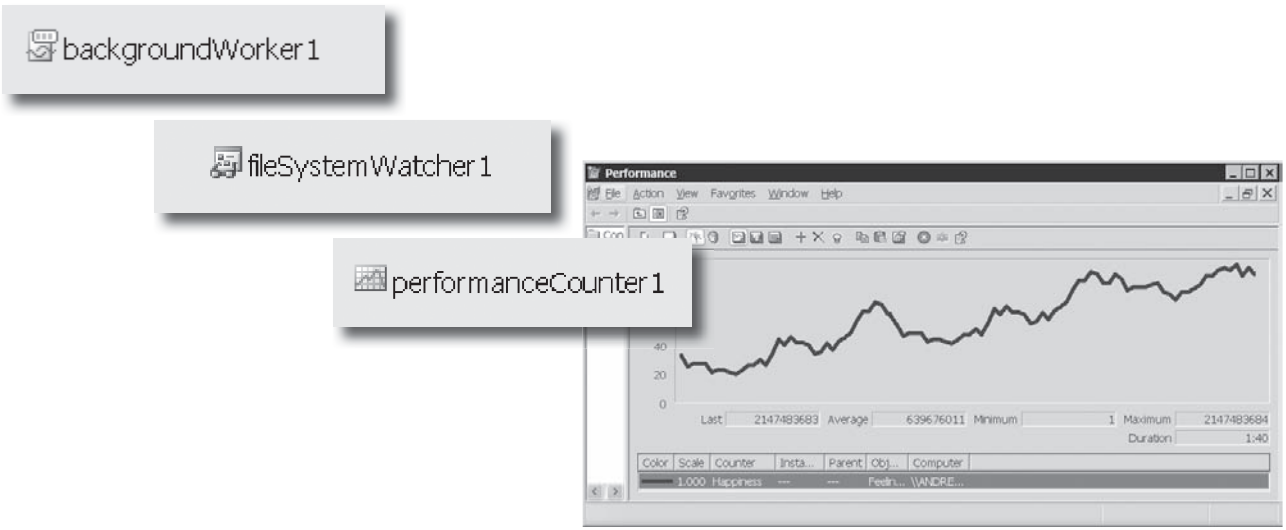
Pozostałości

A 5 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



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 w swoje ręce. Dzięki **Visual Studio IDE** do historii przejdą sytuacje, w których musiałeś pisać jakiś nędzny kod, pozwalający przyciskowi po raz kolejny zadziałać. I to nie wszystko. Dodatkowo będziesz mógł skupić się na **faktycznym wykonywaniu swojej pracy**, zamiast zajmować umysł, pamiętając, który parametr metody odpowiadał za *nazwę* przycisku, a który był odpowiedzialny 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 powtarzalnego kodu.

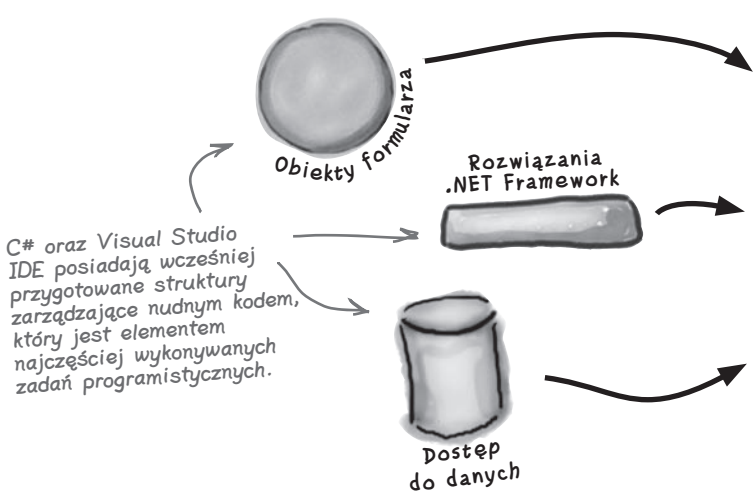
```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
namespace A_New_Program
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
       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

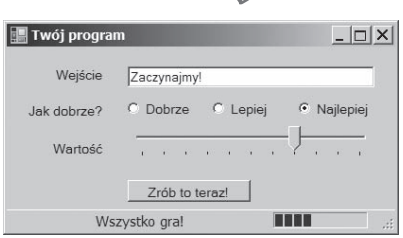
Tyle kodu potrzeba, aby narysować przycisk na formularzu. Dodanie do niego nieco większej ilości elementów wizualnych może spowodować, że wynikowy kod będzie dziesięć razy dłuższy.

Co otrzymujesz razem z Visual Studio oraz C#...

Wraz 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:

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 użytkownika stanie 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 Express, 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 przy pomocy 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.

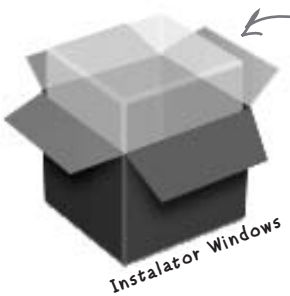
Pomóż dyrektorowi naczelnemu zrezygnować z papieru

Firma papiernicza Objectville Paper 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 czasu jego powrotu powstanie nowy program z książką adresową. W przeciwnym razie… no cóż… nie będzie już starym, dobrym dyrektorem naczelnym.

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 Objectville Paper także chce mieć dostęp do książki adresowej. Chce on używać danych do budowania list kontaktowych i w ten sposób zwiększyć sprzedaż papieru.

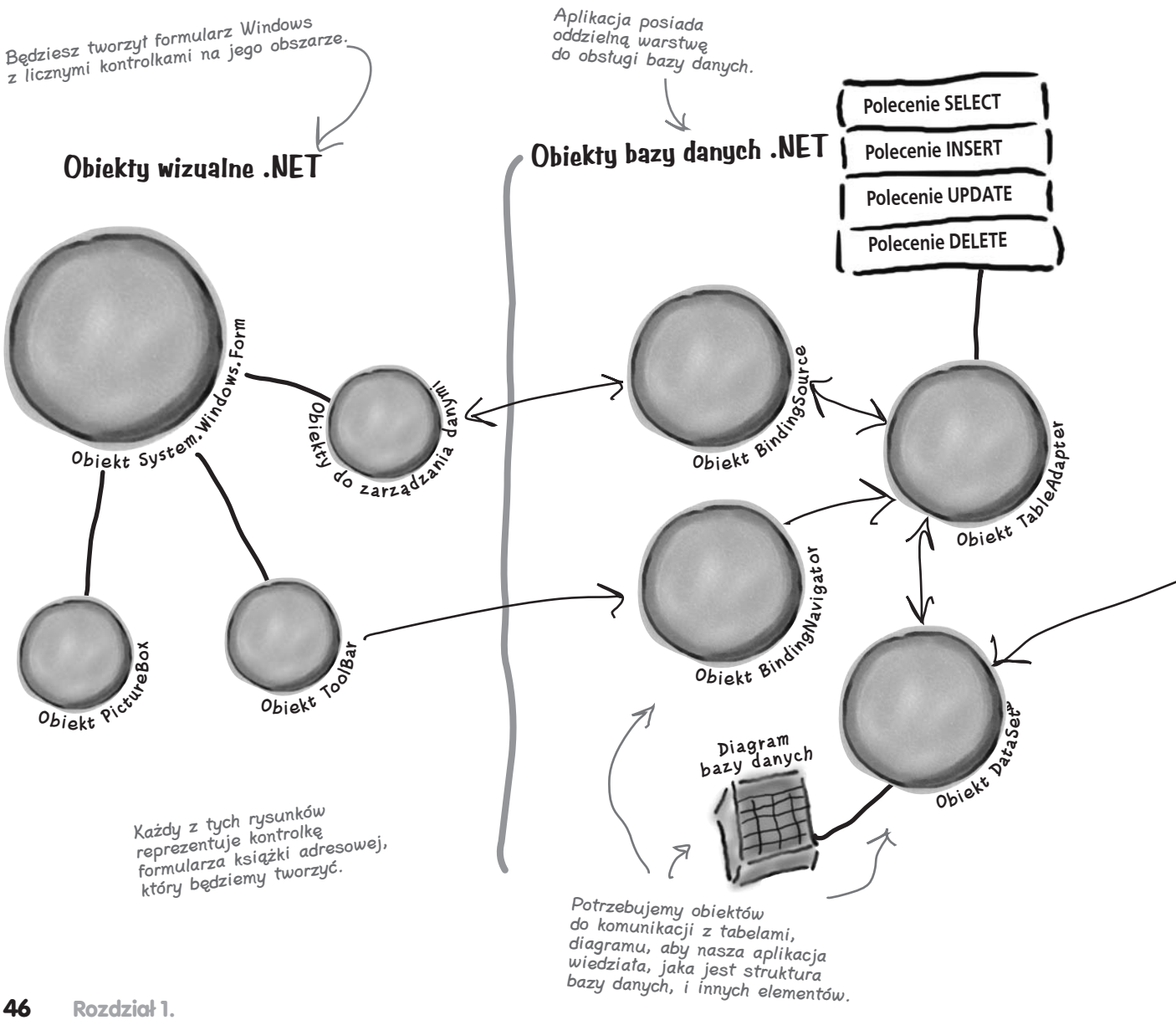
Dyrektor naczelny dochodzi do wniosku, że baza danych będzie najlepszym rozwiązaniem. W ten sposób każdy w firmie będzie miał dostęp do tych informacji, a on będzie musiał zarządzać tylko jedną kopią swoich kontaktów.

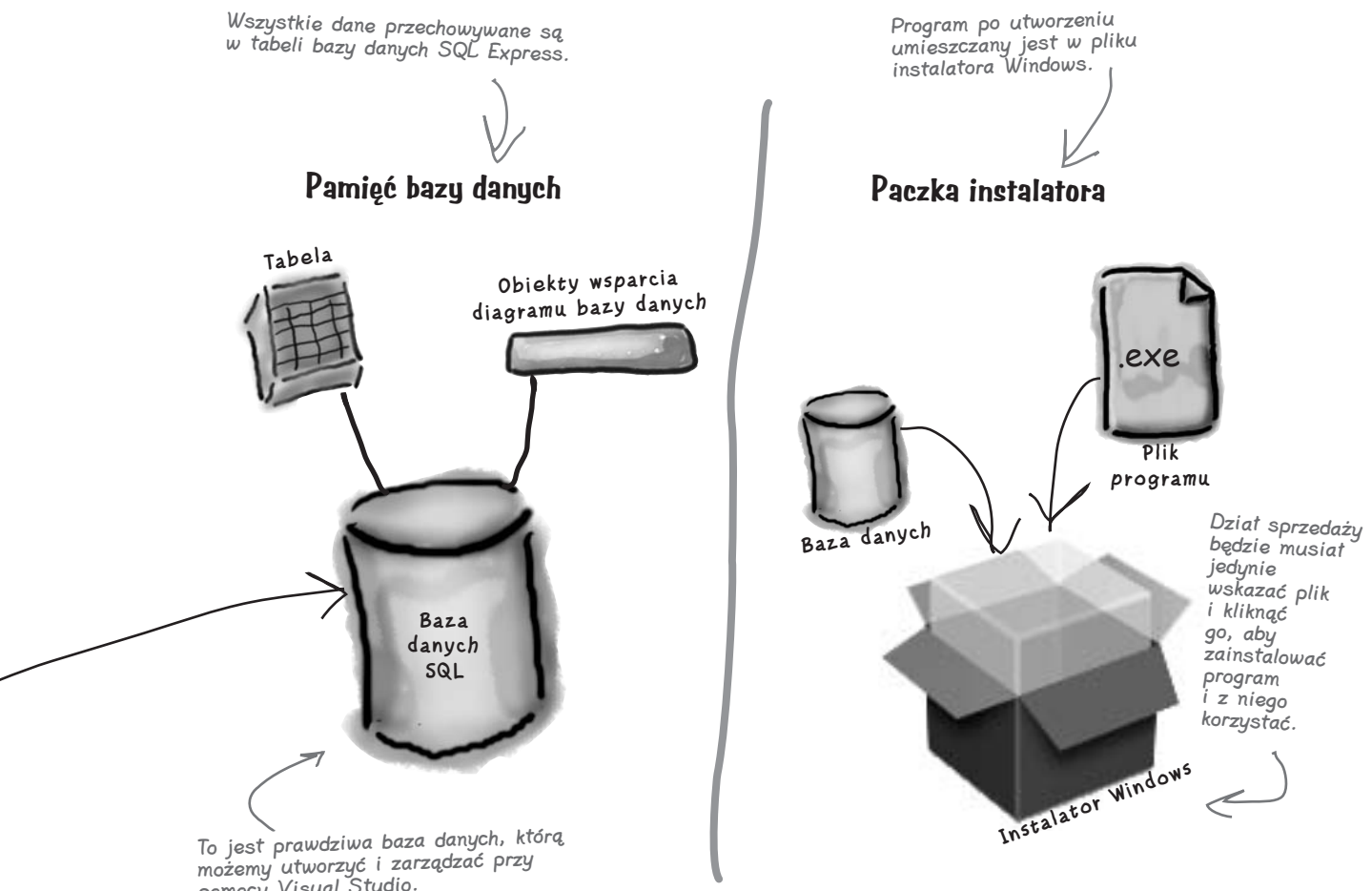
Wiemy już, że Visual Studio ułatwia pracę z bazami danych. Przechowywanie informacji w takiej formie pozwala na jednoczesny dostęp do nich dyrektorowi naczelnemu oraz działowi sprzedaży, mimo że fizycznie istnieje tylko jedna kopia danych.

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.

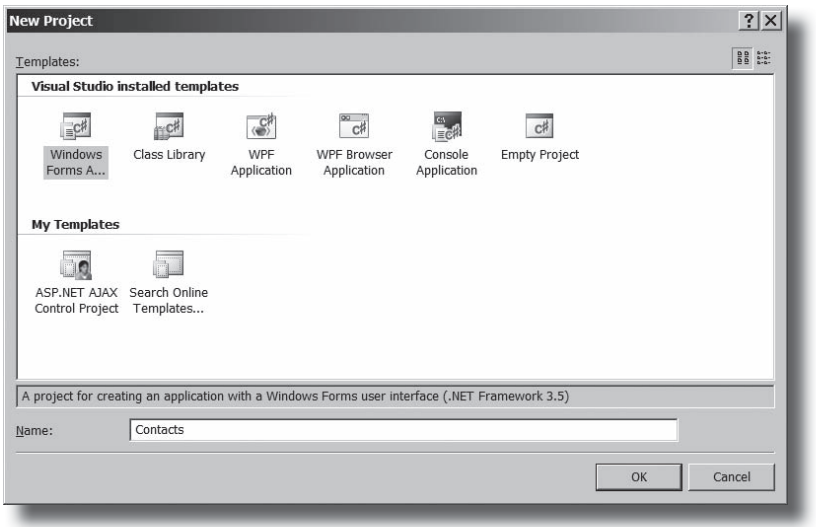
A oto struktura programu, który zamierzamy stworzyć:




Wszystkie dane przechowywane są w tabeli bazy danych SQL Express.
Pamięć bazy danych
Tabela
Obiekty wsparcia diagramu bazy danych
Baza danych SQL
To jest prawdziwa baza danych, którą możemy utworzyć i zarządzać przy pomocy 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 „Contacts" i naciśnij OK.



Obejrzyj to!

Tak wygląda okno „New Project" w Visual Studio 2008 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 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 2008\Projects\Contacts\.

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.

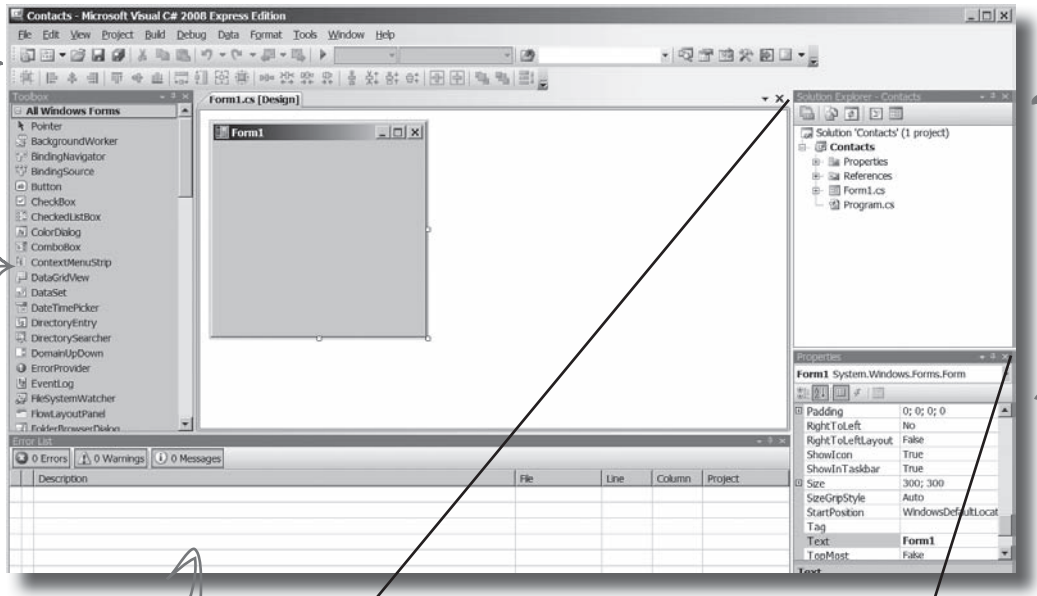
Zaostrz ołówek

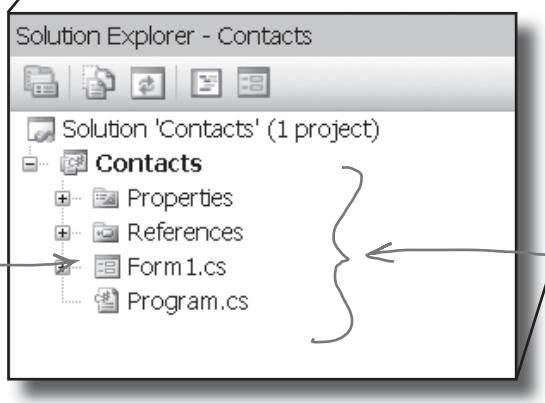
Poniżej pokazany jest wygląd ekranu, jaki prawdopodobnie możesz w tej chwili zaobserwować. Powinieneś odgadnąć, do czego służą poszczególne okna, bazując na wiedzy, którą na tym etapie posiadasz. W każdym z pustych pól wstaw komentarz wyjaśniający przeznaczenie poszczególnych obszarów IDE. Na początek uzupełniliśmy jedno pole.

Ten pasek narzędzi posiada przyciski związane z tym, co jest aktualnie wykonywane przy pomocy 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.





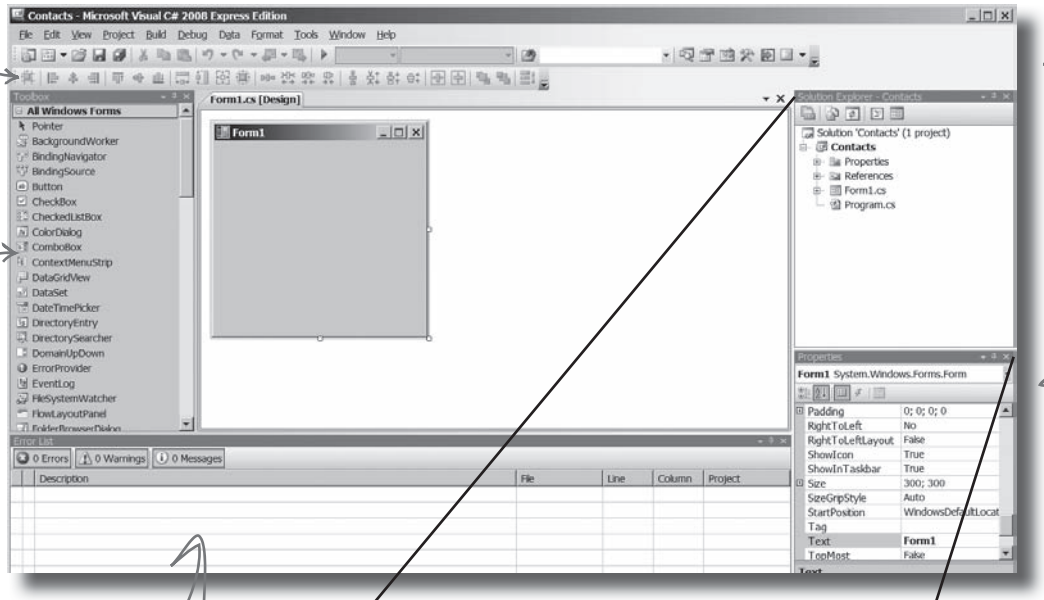
Zaostrz ołówek

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 IDE.

Ten pasek narzędzi posiada przyciski związane z tym, co jest aktualnie wykonywane przy pomocy IDE.

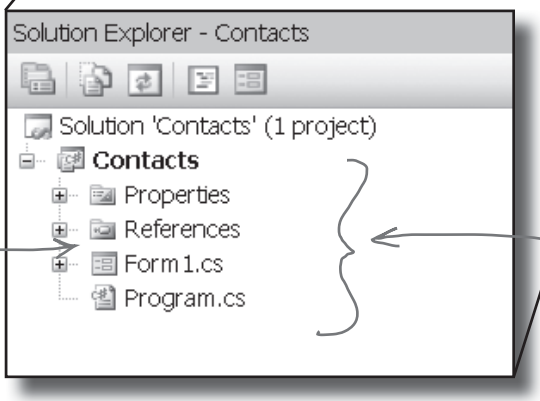
Powiększyliśmy to okno, abyś miał więcej miejsca.



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

To okno pokazuje właściwości wszystkich kontrolek znajdujących się na Twoim formularzu.

Ten obszar na dole służy do debugowania. Pokazuje się wtedy, gdy w Twoim kodzie znajdują się błędy.



Pliki Form1.cs oraz Program.cs zostały utworzone dla Ciebie przez IDE podczas dodawania nowego projektu i pojawiają się w oknie Solution Explorer.

Możesz przełączać się pomiędzy plikami przy pomocy okna Solution Explorer w IDE

Nie ma niemądrych pytań

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 akcji**, czyli tego, który wykonuje całą pracę.

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

O: Po pierwsze, musisz używać Visual Studio 2008 — w Visual Studio 2005 projekty przechowywane są w innym miejscu. Kiedy zaczynasz tworzyć nowy projekt w Visual Studio 2008, 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 2008\Projects. Próbując otworzyć nowy projekt lub zamknąć tymczasowy, zostaniesz poproszony o wybór jednej z dwóch opcji — zapisania lub odrzucenia zmian.

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 Microsoft). 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, możesz zmienić każdą część swojego programu. IDE jest skonfigurowane w taki sposób, że nazywa nowe pliki z zachowaniem ostrożności. Gdy dodajesz plik do projektu, jego nazwa ma wpływ na część generowanego kodu, który ją zawiera. W niektórych przypadkach wprowadzenie nowej nazwy pliku pociąga za sobą konieczność wprowadzenia zmian we fragmentach kodu lub powoduje dyskomfort życia z różnymi nazwami pliku oraz klasy w nim zapisanej. W związku z tym, że jest to niepożądane, zalecamy pozostawienie nazw, chyba że istnieje naprawdę poważny powód, aby dokonać takiej zmiany.

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. Twój ekran powinien wyglądać dokładnie tak samo, jak inne w tym rozdziale.

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

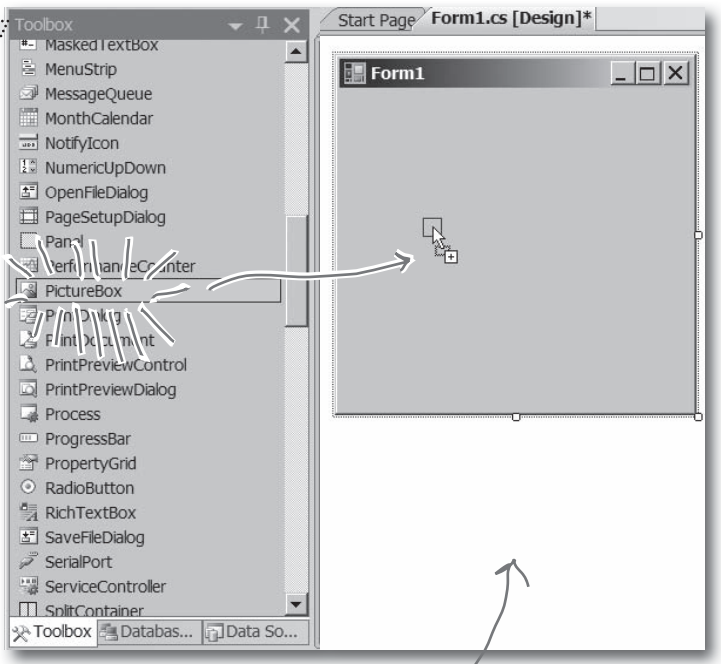
Upewnienie się, że aplikacja robi dokładnie to, do czego została stworzona, należy wyłącznie do Ciebie.

Stwórz interfejs użytkownika

Dodawanie kontrolek i nadawanie blasku interfejsowi użytkownika jest tak proste, jak przeciąganie i upuszczanie 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.

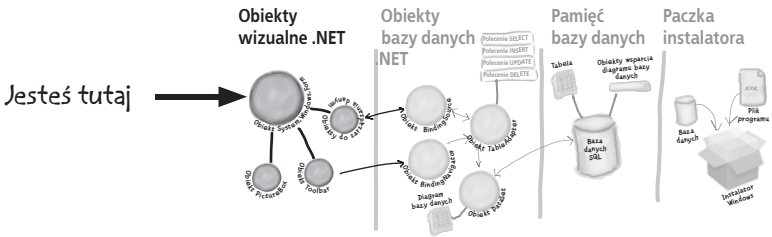


Za każdym razem, gdy wprowadzasz zmiany we właściwościach kontrolki formularza, kod w pliku Form1.Designer.cs jest zmieniany przez IDE.

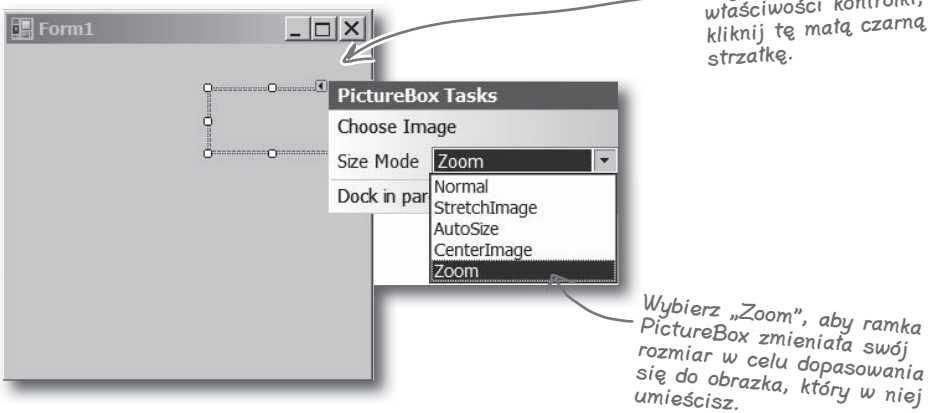
Form1.Designer.cs

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

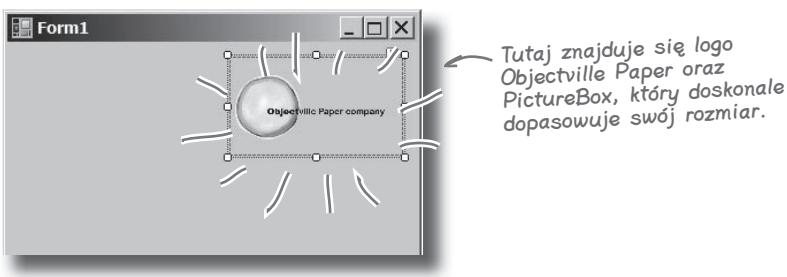
W swoim czasie powiemy znacznie więcej o tworzeniu dobrych interfejsów użytkownika. Na tym etapie wystarczy, że wstawisz logo oraz inne kontrolki do formularza i zadbasz o jego prawidłowe ***zachowanie***. O stylowy wygląd zatroszczymy się później.



2 **Ustaw tryb Zoom w 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.



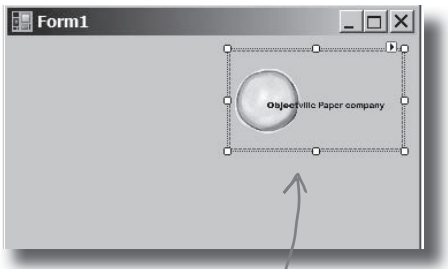
3 **Pobierz logo firmy Objectville Paper.**
Pobierz logo Objectville Paper z serwera ftp (*ftp://ftp.helion.pl/przyklady/hfcsh.zip*) i zapisz je na twardym dysku. Następnie wybierz strzałkę właściwości i kliknij Choose Image. Zaznacz pole wyboru Local resource, a następnie kliknij Import i znajdź Logo. Wszystko powinno ustawić się poprawnie.



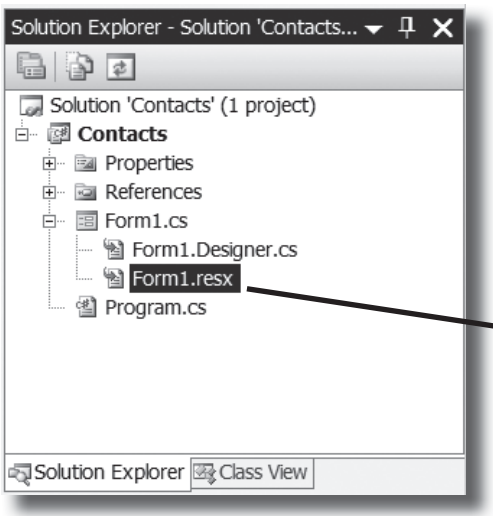
Visual Studio za kulisami

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 zasoby i skojarzyło je 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.

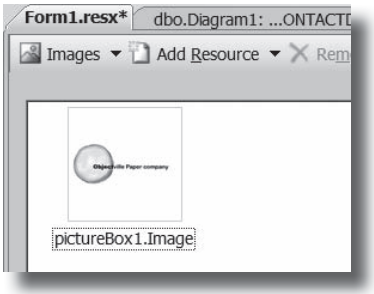
Kiedy przeciągasz kontrolkę PictureBox na formularz, IDE automatycznie tworzy plik Form1.resx, aby przechowywać zasoby i trzymać je razem w projekcie. Kliknij dwukrotnie ten plik, a zobaczysz niedawno zaimportowany obrazek.

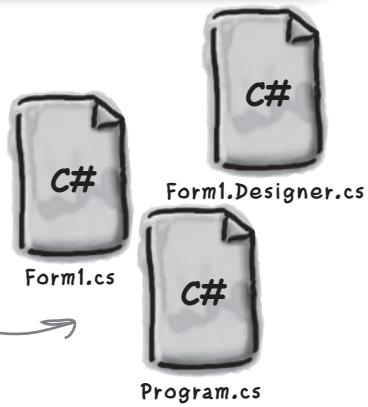


Ten obrazek jest teraz w zasobach aplikacji listy kontaktowej.



Gdy klikniesz Form1.resx, w oknie Solution Explorer możesz zobaczyć zaimportowane logo. Plik łączy obrazek z kontrolką PictureBox, natomiast IDE dodaje odpowiedni kod, niezbędny do takiego powiązania.





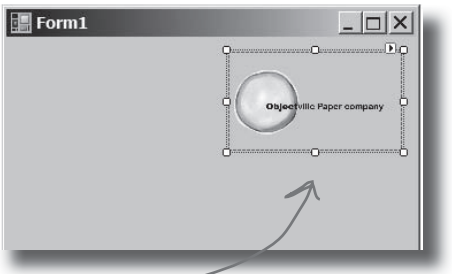
Tu są pliki, które Visual Studio wygenerowało wcześniej.

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 kodu generowanego automatycznie

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 informacje o programie za każdym razem, gdy użytkownik je kliknie.

Upewnij się, że IDE wyświetla formularz, a następnie kliknij dwukrotnie kontrolkę PictureBox. Powinieneś zobaczyć fragment pojawiającego się kodu podobny do tego:




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

    private void pictureBox1_Click (object sender, EventArgs e)
    {

    }
}
```

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

Nazwa tej metody prawidłowo sugeruje czynność wykonywaną podczas jej wywołania, czyli kliknięcie kontrolki 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 pojawianie się okna z komunikatem w postaci wprowadzonego tekstu. Okno będzie miało tytuł „O programie".

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

Nie ma niemądrych pytań

P: Co to jest metoda?

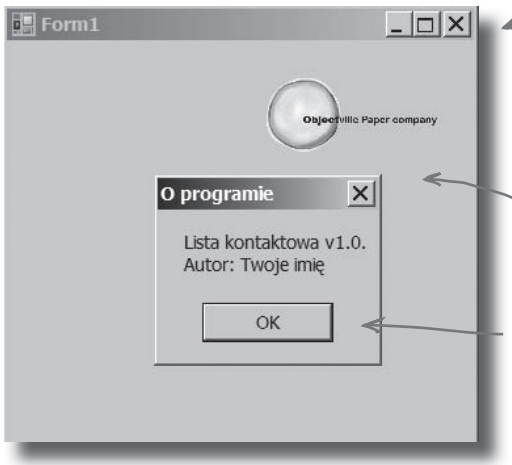
O: Metoda to po prostu *nazwany fragment kodu*. Powiemy sobie na ten temat znacznie więcej w rozdziale 2.

P: Co robi jakieś \n w kodzie?

O: To symbol łamania linii. W ten sposób mówimy C#, aby umieścił fragment „Lista kontaktowa v1.0." w jednej linii, a następnie rozpoczął nową dla pozostałej części.

Możesz już 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 Objectville Paper powoduje pokazanie się okna dialogowego, które właśnie napisałeś.

Gdzie są moje pliki?

Kiedy uruchamiasz swój program, Visual Studio kopiuje wszystkie pliki do katalogu *Moje Dokumenty\Visual Studio 2008\Projects\Contacts\Contacts\bin\Debug*. Możesz nawet przejść do niego i uruchomić poprzez podwójne kliknięcie myszką 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 ma niemądrych pytań

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.

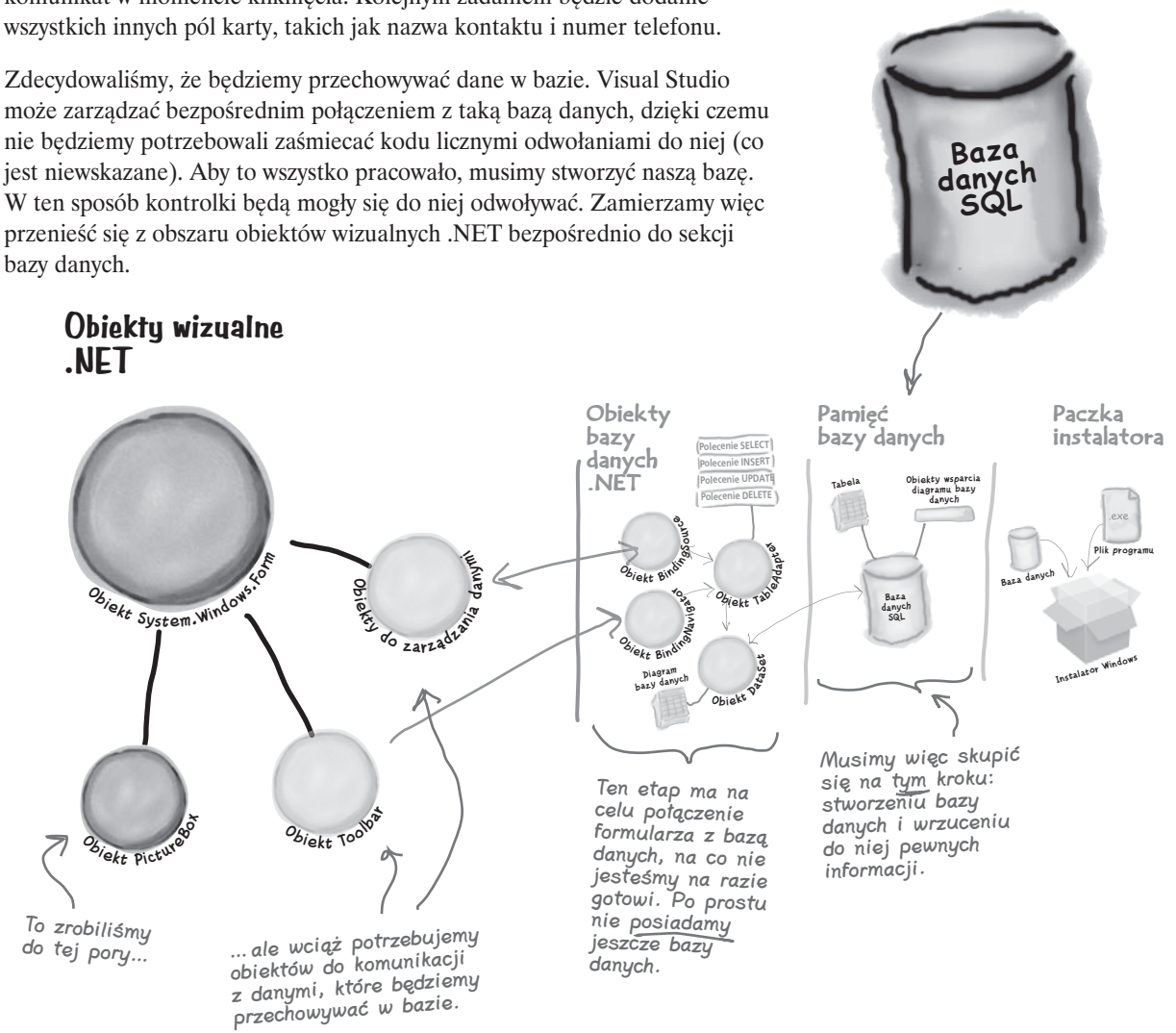
P: Nie widzę żadnego przycisku „Stop Debugging" na moim pasku narzędzi. O co chodzi?

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 przycisk się pojawił.

Co zrobiliśmy do tej pory

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

Zdecydowaliśmy, że będziemy przechowywać dane w bazie. Visual Studio może zarządzać bezpośrednim połączeniem z taką bazą danych, dzięki czemu nie będziemy potrzebowali zaśmiecać kodu licznymi odwołaniami do niej (co jest niewskazane). Aby to wszystko pracowało, musimy stworzyć naszą bazę. W ten sposób kontrolki będą 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

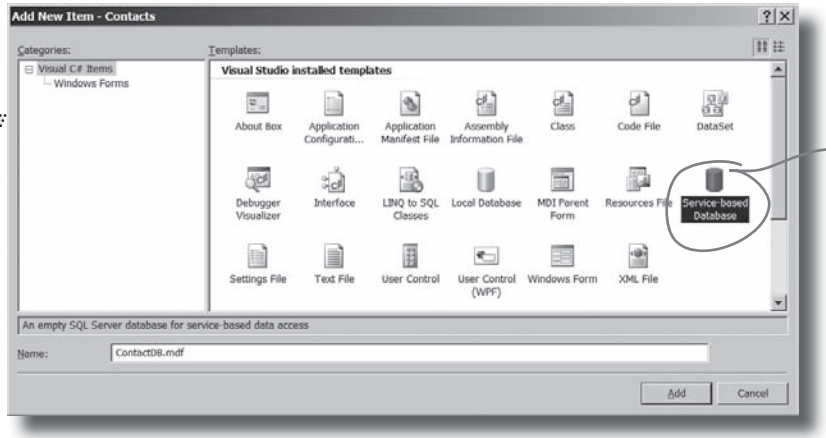
Upewnij się, że skończyłeś debugować, zanim przejdziesz dalej.

Zanim dodamy resztę pól do formularza, musimy stworzyć bazę, do której będzie się on odnosił. IDE może wygenerować spory kawał 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 Contacts**, wybierz Add, a następnie New Item. Kliknij ikonę SQL Database, a jako nazwę wpisz ContactDB.mdf.

Ten plik jest naszą nową bazą danych.

ContactDB.mdf



Wybierz ikonę właściwą dla Twojej wersji — **SQL Database,** jeżeli używasz Visual Studio Express 2005, lub **Service-Based Database**, jeśli korzystasz z wersji z roku 2008.

Baza danych SQL może pracować tylko wtedy, gdy zainstalowałeś wcześniej SQL Server Express. Cofnij się do wstępu, jeżeli nie wiesz, jak to zrobić.

2. **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.

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

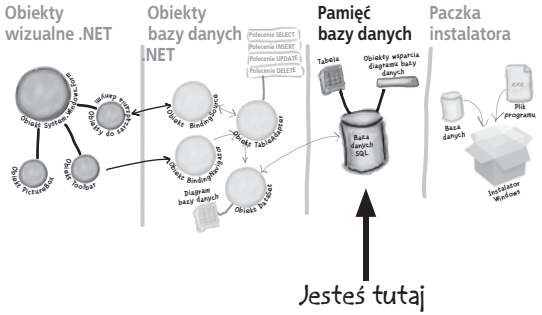
Obejrzyj to!

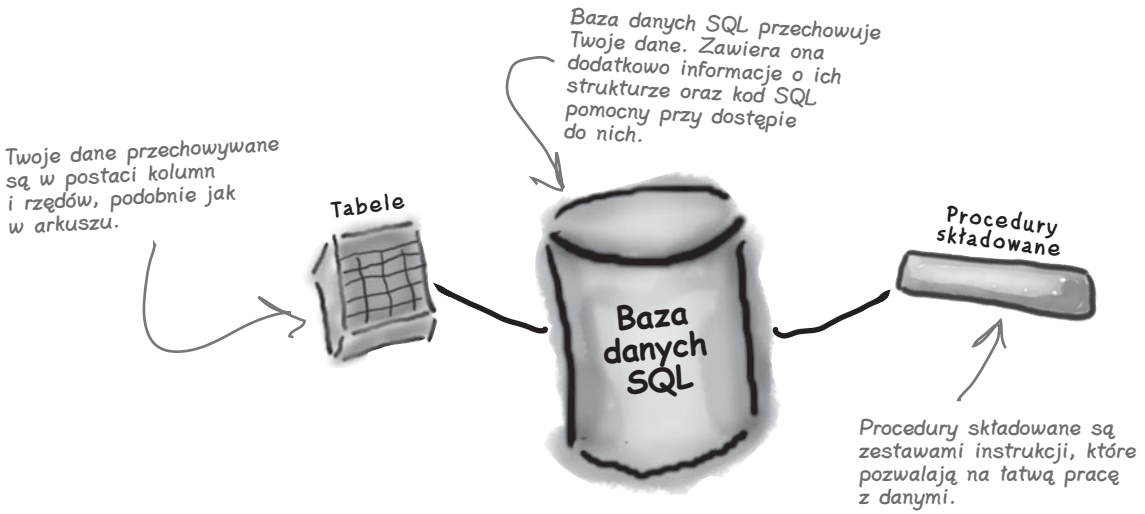
Visual Studio 2008 Professional oraz Team Foundation nie posiadają okna Database Explorer. Zamiast tego mają okno Server Explorer, które potrafi zrobić dokładnie to samo, a oprócz tego umożliwia przeglądanie danych w sieci.

IDE stworzyło bazę danych

Kiedy nakazałeś IDE dodać nową bazę SQL do projektu, IDE ją stworzyło. **Baza danych SQL** to system, który przechowuje dane w sposób zorganizowany i relacyjny. Dodatkowo IDE udostępnia wszystkie narzędzia niezbędne do pracy z nimi.

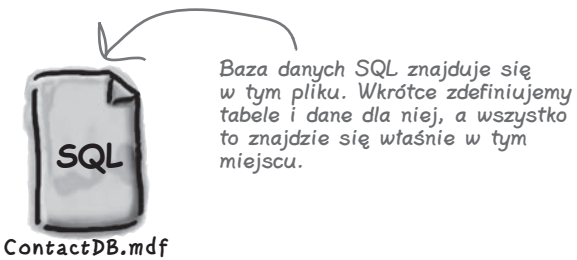
Dane w bazie SQL przechowywane są w tabelach. Na bieżące potrzeby wystarczy wyobrazić sobie, że baza to rodzaj arkusza. Organizuje on dane w ramach wierszy i kolumn. Kolumny reprezentują ich kategorie, takie jak nazwa kontaktu i numer telefonu, natomiast każdy rząd reprezentuje jedną kartę kontaktową.





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.

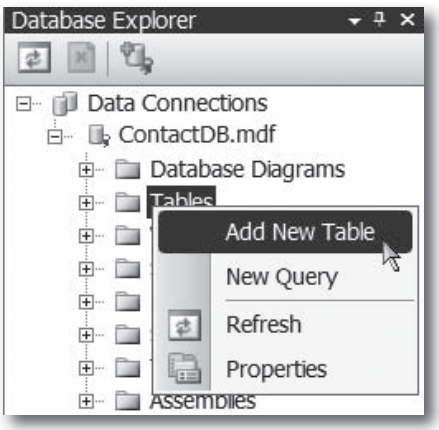


[Notka działu marketingu: Czy możemy wstawić tutaj reklamę „Head First SQL"?]

Tworzenie tabeli dla listy kontaktowej

Mamy już bazę danych. Potrzebujemy jeszcze przechowywać 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 Add New Table. Spowoduje to otwarcie okna, w którym możesz zdefiniować nazwy kolumn nowo 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 pole Allow Nulls jest odznaczone.

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

To jest przycisk Set Primary Key. Klucz główny pomaga bazie danych przyspieszać procedury wyszukiwania rekordów.

Nie ma niemądrych pytań

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. Pola takie zawsze będą miały typ, taki jak String, Date lub Bool.

P: Do czego jest nam potrzebna kolumna ContactID?

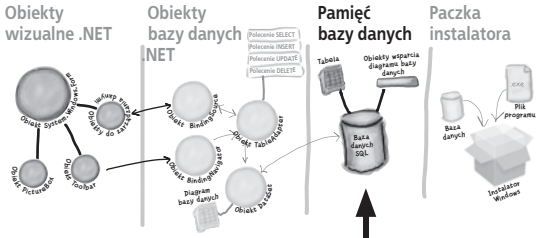
O: Umożliwia ona posiadanie unikalnego identyfikatora przez każdy rekord w większości tabel. Dlatego w przypadku przechowywania listy kontaktów do poszczególnych osób zdecydowaliśmy się stworzyć specjalną kolumnę i nazwać ją ContactID.

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. Skup się na podstawowych pojęciach. Poświęcimy więcej czasu na tematy związane z bazami danych w kolejnych rozdziałach tej książki. Jeśli umierasz z ciekawości, zawsze możesz sięgnąć po ***Head First SQL*** i czytać ją równolegle z tą pozycją.

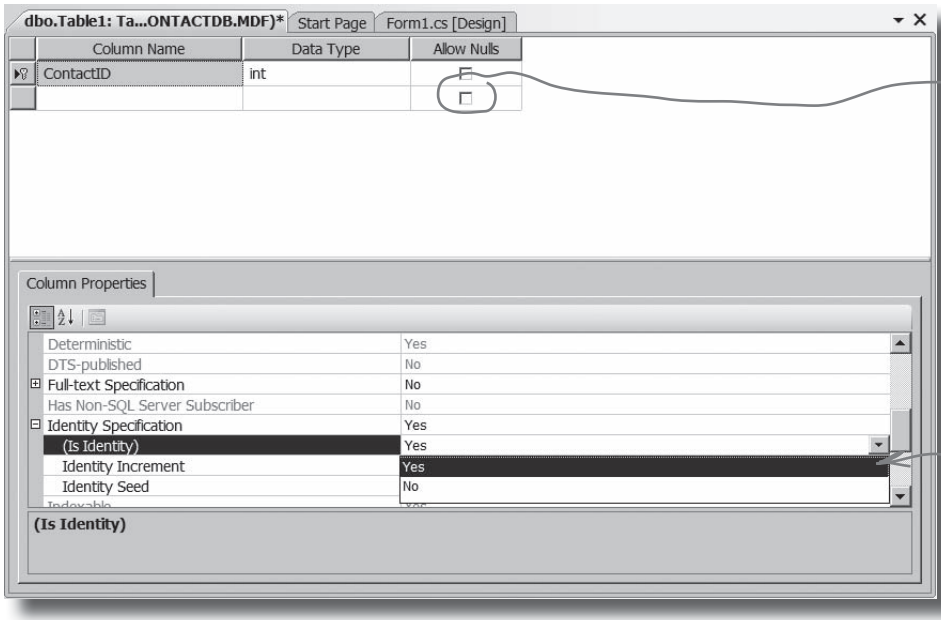


3 **Powiedz bazie danych, aby automatycznie generowała identyfikatory.**

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.

Przewiń zawartość okna w dół, aż do odnalezienia we właściwościach poniżej tabeli pola Identity Specification. Kliknij przycisk + i wybierz Yes w polu (Is Identity).

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



Ważne jest, abyś pozostawił to pole odznaczone. Dopóki klucz główny jest podstawowym sposobem wyszukiwania rekordów, dopóty pole ContactID musi mieć wartość.

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

Pola na karcie kontaktowej stają się kolumnami w tabeli People

Teraz, kiedy już stworzyłeś 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.

Chcemy przechowywać dane każdej osoby znajdującej się na liście kontaktowej. Należą do nich: nazwa kontaktu, nazwa firmy, numer telefonu, adres e-mail, wartość określająca, czy jest to klient firmy, oraz data ostatniego kontaktu telefonicznego.

Każde pole na karcie adresowej musi mieć swój odpowiednik w tabeli People.

WYTĘŻ UMYSŁ

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

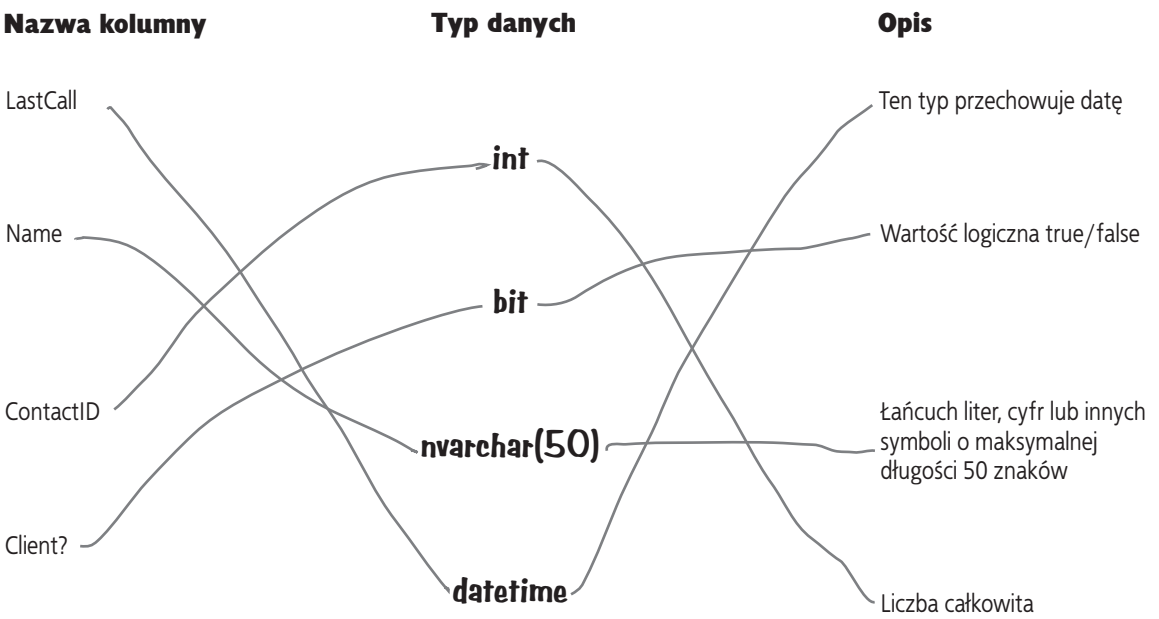
KTO CO ROBI?

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
	int	
Name		Wartość logiczna true/false
	bit	
ContactID		Łańcuch liter, cyfr lub innych symboli o maksymalnej długości 50 znaków
	nvarchar(50)	
Client?		Liczba całkowita
	datetime	

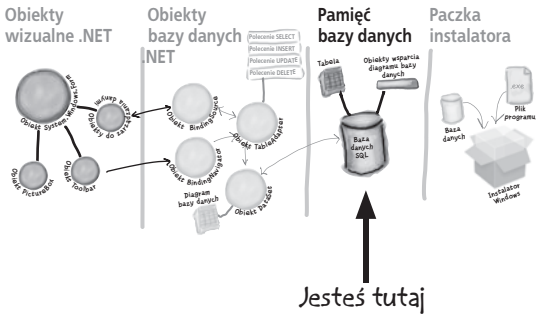
KTO CO ROBI?

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.



Zakończ tworzenie tabeli

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



dbo.People: T...CONTACTDB.MDF) | People: Query...CONTACTDB.MDF)

Column Name	Data Type	Allow Nulls
ContactID	int	☐
Name	nvarchar(50)	☑
Company	nvarchar(50)	☑
Telephone	nvarchar(50)	☑
Email	nvarchar(50)	☑
Client	bit	☑
LastCall	datetime	☑
		☐

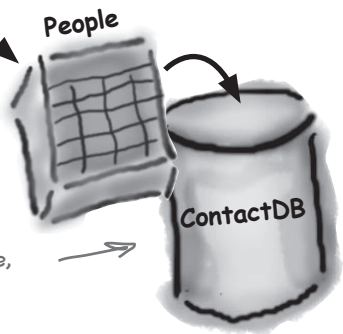
Pola bitowe przechowują wartości True lub False i mogą być reprezentowane przez kontrolkę CheckBox.

Jeżeli odznaczysz pole Allow Nulls, kolumna będzie musiała posiadać wartość.

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

Kliknij przycisk Save na pasku narzędzi, a tabela zostanie zapisana. Zostaniesz poproszony o podanie jej nazwy. Wpisz „People” i naciśnij OK.

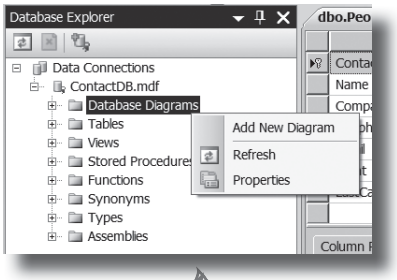
Za każdym razem, gdy mówiliśmy o tabeli „People”, mieliśmy na myśli tę tabelę, jednak dopiero teraz uzyskała ona swoją oficjalną nazwę.



Tutaj tworzymy tabelę People, która zostanie umieszczona w bazie danych ContactDB.

Utwórz diagram dla swoich danych, aby aplikacja miała do nich dostęp

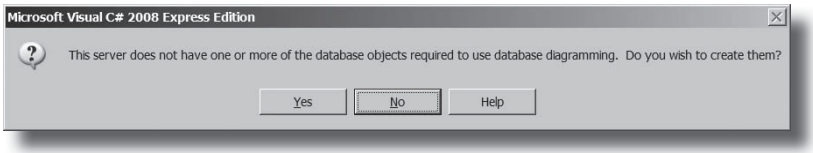
Stworzyłeś bazę danych i tabelę. Nadszedł już czas, aby aplikacja się o tym dowiedziała. To jest właśnie moment, w którym diagram wkracza do akcji. **Diagram bazy danych** jest prostym opisem tabeli, którego Visual Studio może użyć do pracy z nią. Pozwala on także IDE automatycznie generować instrukcje SQL używane przy dodawaniu, modyfikacji oraz usuwaniu rekordów.



Pamiętaj, że opcje te są wywoływane dla bazy ContactDB i dotyczą właśnie tej konkretnej bazy danych.

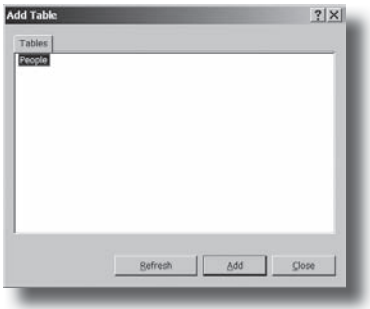
1 **Stwórz nowy diagram bazy danych.**
Przejdź do okna Database Explorer i kliknij prawym przyciskiem myszy element Database Diagrams. Wybierz Add New Diagram.

2 **Pozwól IDE wygenerować kod dostępu.**
Zanim IDE dowie się o konkretnej tabeli, musi najpierw wygenerować kilka podstawowych procedur składowanych do komunikacji z bazą. Po prostu naciśnij Yes i pozwól IDE wykonać całą pracę.

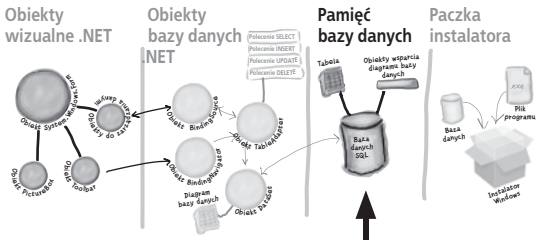


IDE tworzy kilka procedur składowanych, które pozwolą na interakcję kodu programu z utworzoną bazą danych.

3 **Wybierz tabele, z którymi chcesz pracować.**
Z okna, które powinno się pojawić, wybierz tabelę People i kliknij Add. W tym momencie IDE jest gotowe do wygenerowania kodu specyficznego dla danej tabeli.



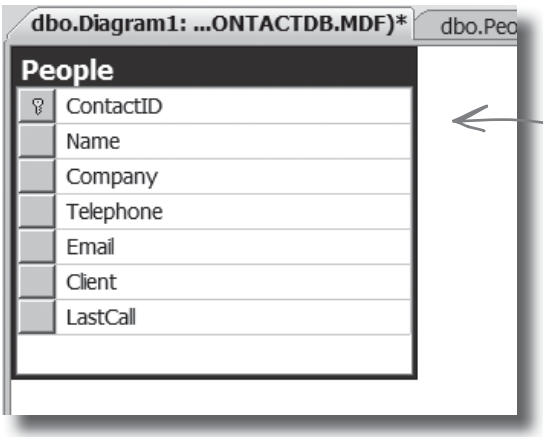
Jeśli posiadasz bazę danych z wieloma tabelami, to każda z nich jest reprezentowana przez element w tym oknie.



4 **Nazwij diagram PeopleDiagram.**
Wybierz File/Save. Zostaniesz poproszony o podanie nazwy dla nowego diagramu bazy danych. Aby dopełnić formalności, nazwij go PeopleDiagram.

Jeżeli używasz Visual Studio 2005, wybierz zamiast tego File/Save All.

Diagram bazy danych pokazany jest tutaj. Jest to bardzo prosta reprezentacja twojej tabeli.



Jest to jedynie ilustracja projektu bazy danych, który utworzyłeś. Wyświetlane są tutaj wszystkie kolumny tabeli, a ContactID jest oznaczona jako klucz główny.

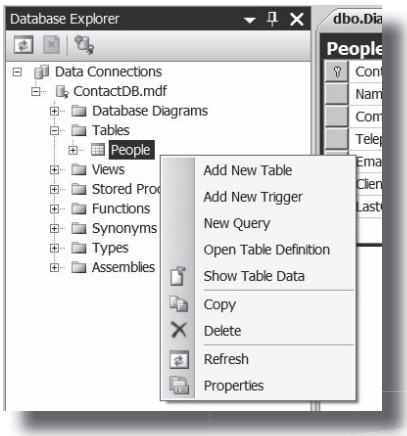
Jeśli masz w bazie danych jakieś inne tabele, które chciałeś dodać do diagramu, to one także się tutaj pojawią.

Diagram bazy danych jest dla Visual Studio IDE opisem tabeli. IDE używa go do automatycznego generowania kodu pomocnego w pracy z bazą.

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 to 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: *Liz Nelson*
Objectville Paper company
Firma: *JTP*
Telefon: *(419) 555-2578*
E-mail: *LizNelson@JTP.org*
Klient: *Tak* **Ostatni kontakt:** *04.05.2006*

Nazwa: *Lucinda Ericson*
Objectville Paper company
Firma: *Ericson Events*
Telefon: *(212) 555-9523*
E-mail: *Lucy@EricsonEvents.info*
Klient: *Nie* **Ostatni kontakt:** *17.05.2007*

Nazwa: *Lloyd Jones*
Objectville Paper company
Firma: *Black Box Inc.*
Telefon: *(718) 555-5638*
E-mail: *LJones@xblackboxinc.com*
Klient: *Tak* **Ostatni kontakt:** *26.05.2007*

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 zwyktego „Save", zapisującego tylko plik, nad którym w danej chwili się pracuje.

Nie ma niemądrych pytań

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.mdf SQL Server Express. 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.mdf znajdującym się w jej katalogu wyjściowym.

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

ContactDB.mdf

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.

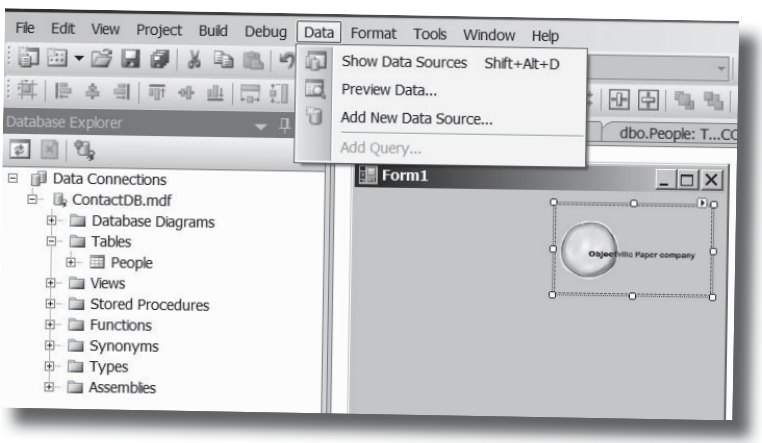
1 **Przejdź z powrotem do formularza aplikacji.**
Zamknij tabelę People i diagram bazy danych ContactDB. Powinieneś w tej chwili widzieć zakładkę Form1.cs [Design].

Musisz zamknąć zarówno tabelę z danymi, jak i diagram, aby powrócić do widoku formularza.

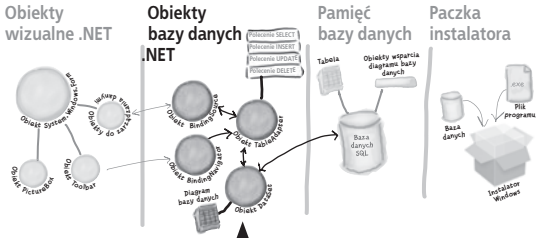
People: Query...CONTACTDB.MDF) | dbo.People: T...CONTACTDB.MDF) | Start Page | Form1.cs | Form1.cs [Design]

	ContactID	Name	Company	Telephone	Email	Client	LastCall
▶	1	Lloyd Jones	Black Box Inc	718555638	ljones@blackboxin...	True	2007-05-26 1...
	2	Lucinda Erics...	Ericson Events	2125559523	Lucy@ericsoneven...	False	2007-05-17 1...
	3	Liz Nelson	JTP	4195552578	LizNelson@JTP.org	True	2006-03-04 0...
	4	Matt Franks	XYZ Industries	21255581125	Matt.Franks@xyzin...	True	2007-05-26 0...
	5	Sarah Kalter	Kalter, Riddle, and ...	6145555641	Sarah@KRS.org	False	2006-12-10 0...
	6	Laverne Smith	XYZ Industries	2125558129	Laverne.Smith@xy...	True	2007-05-26 0...
*	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 miało za zadanie obsługę komunikacji pomiędzy formularzem i bazą danych.

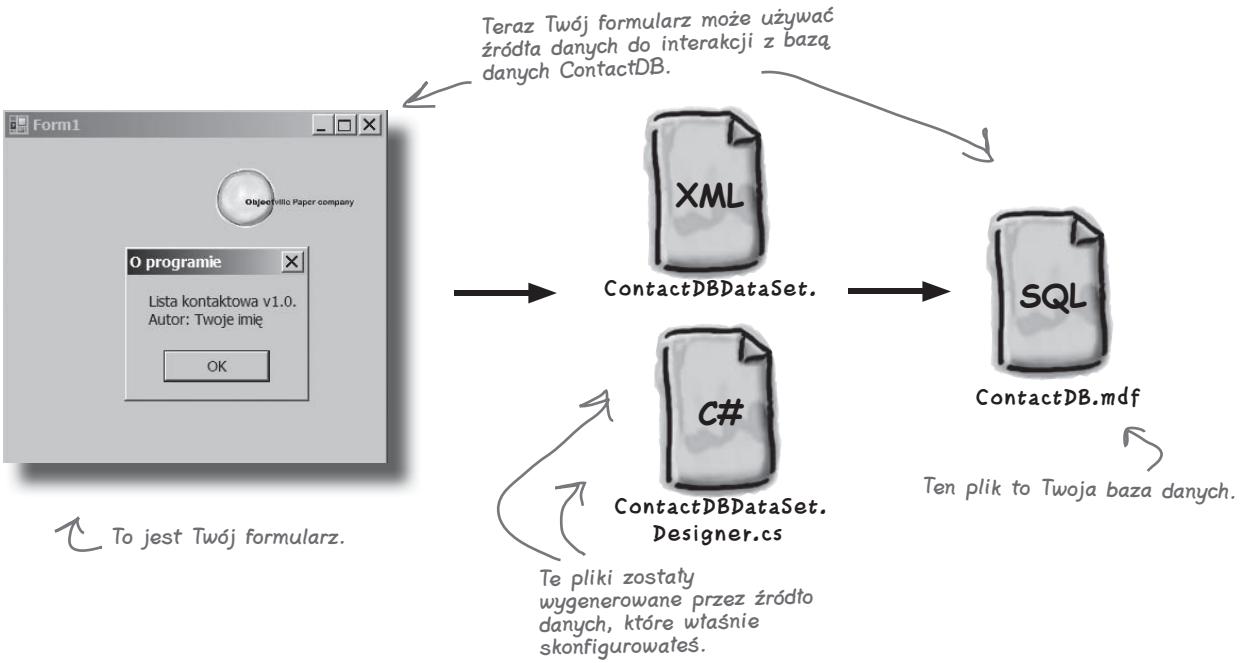


3 **Skonfiguruj źródło danych.**

Teraz musisz dostosować źródło danych, aby używało bazy ContactDB. W tym celu powinieneś wykonać następujące czynności:

- wybrać Database i kliknąć przycisk Next,
- kliknąć Next na ekranie „Choose Your Data Connection”,
- upewnić się, że pole wyboru „Yes, save the connection as:” na ekranie „Save the Connection” jest zaznaczone, i nacisnąć Next,
- na ekranie „Choose Your Objects” zaznaczyć pole przy elemencie Tables,
- upewnić się, że w polu DataSet name wpisane jest „ContactDBDataSet” i kliknąć Finish.

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



Dodaj kontrolki powiązane z bazą danych do formularza

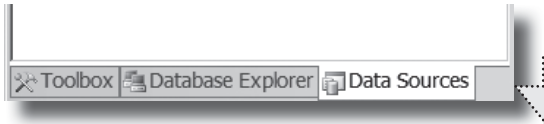
Wymaga to nieco wysiłku, ale wracamy tutaj, aby utworzyć obiekty formularza, które będą współpracować z bazą danych.

Teraz możemy powrócić do naszego formularza i dodać kilka kontrolek. Nie będą to jednak zwykłe kontrolki, tylko takie, które są *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:

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

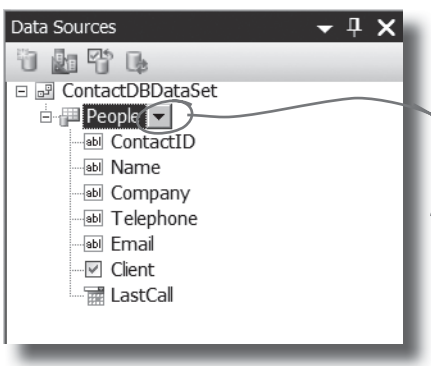
1 **Wybierz źródło danych, którego chcesz używać.**
Wybierz Show Data Sources z menu Data. Spowoduje to pojawienie się okna Data Sources z wyświetlonymi źródłami danych skonfigurowanymi dla tej aplikacji.



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

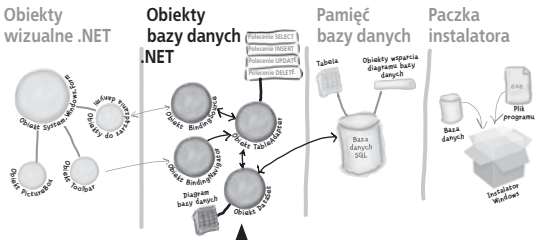
To okno pokazuje wszystkie Twoje źródła danych. W tej chwili skonfigurowane jest tylko jedno, ale oczywiście nie ma żadnych przeciwwskazań do posiadania większej ilości dla innych tabel lub baz danych.

2 **Wybierz tabelę People.**
Pod elementem ContactDBDataSet powinieneś zobaczyć tabelę People i wszystkie jej kolumny. Jeżeli ich nie widzisz, kliknij strzałkę menu rozwijalnego i wybierz Details.



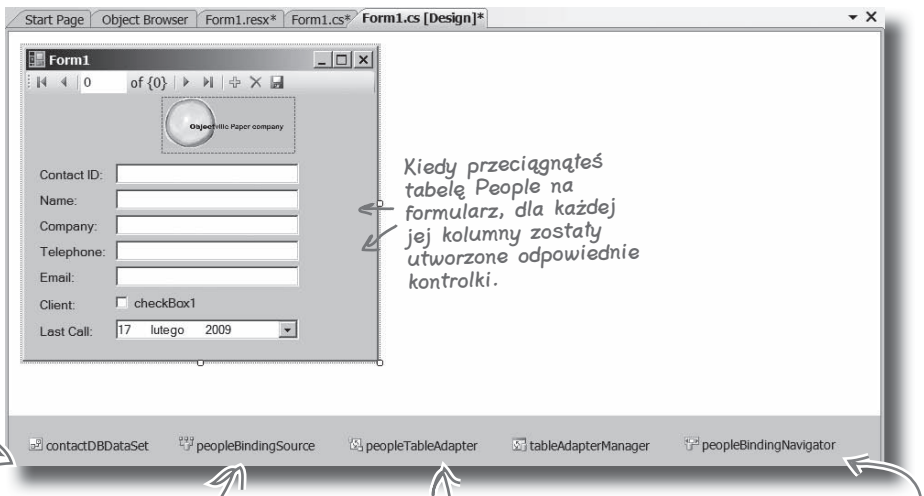
To jest mała strzałka, w którą powinieneś kliknąć.

Wszystkie pola, które utworzyłeś, powinny się tutaj pojawić.



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 zbiór danych, który został utworzony przez IDE do zarządzania tabelą People oraz bazą danych ContactDB.

Ten obiekt łączy formularz z tabelą People bazy danych.

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

Obiekt peopleBindingNavigator łączy kontrolki paska narzędzi z tabelą bazy danych.

Dobre programy są intuicyjne w użyciu

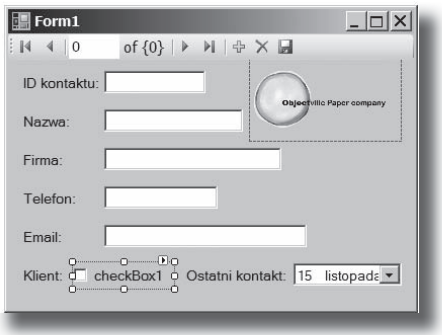
Co prawda formularz teraz działa, ale nie wygląda dobrze. Twoja aplikacja musi być więcej niż tylko funkcjonalna. Powinna być łatwa w użyciu. Dzieli Cię zaledwie kilka etapów od osiągnięcia wyglądu formularza zbliżonego do wyglądu papierowych kart, których używaliśmy na początku niniejszego rozdziału.

Nasz formularz będzie bardziej intuicyjny, jeżeli będzie wyglądał jak karta kontaktowa.

① Wyrównaj kontrolki i pola tekstowe.

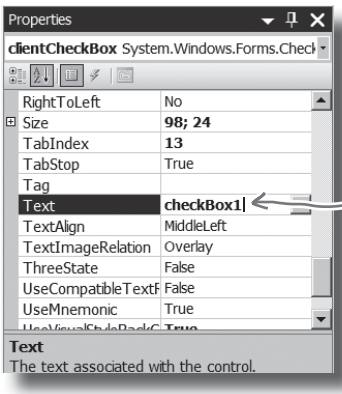
Wyrównaj kontrolki i pola tekstowe względem lewej krawędzi formularza. Będzie on wyglądał podobnie do innych aplikacji, a użytkownicy będą czuli się znacznie bardziej komfortowo podczas korzystania z programu. Przetłumacz angielskie treści etykiet, powstałych automatycznie na podstawie nazw kolumn z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

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

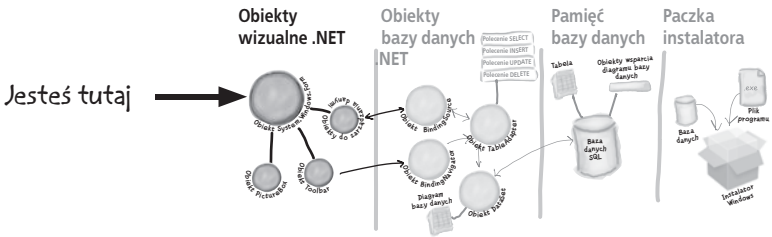


② Zmień właściwość Text kontrolki CheckBox.

Podczas wstawiania kontrolek do formularza obok kontrolki Checkbox pojawia się pole tekstowe, które musi być usunięte. 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.

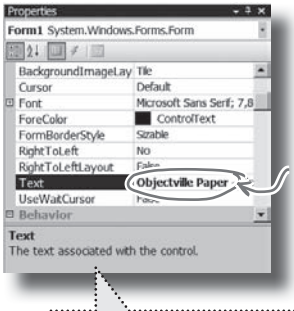


③ **Nadaj aplikacji profesjonalny wygląd.**
Możesz także zmienić nazwę formularza. Dokonasz tego poprzez kliknięcie w dowolnym jego miejscu i odnalezienie właściwości Text w oknie Properties. Dla naszych potrzeb zmień nazwę na „Objectville Paper Co. — Lista Kontaktów”.

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 to 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.

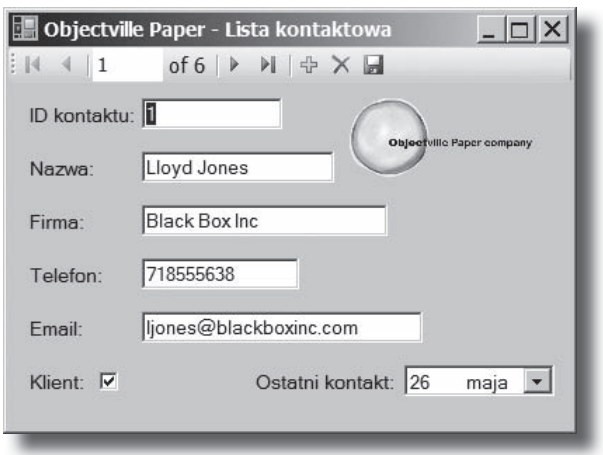
Jazda próbna

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.



Poświęcimy na to więcej czasu w następnym rozdziale.

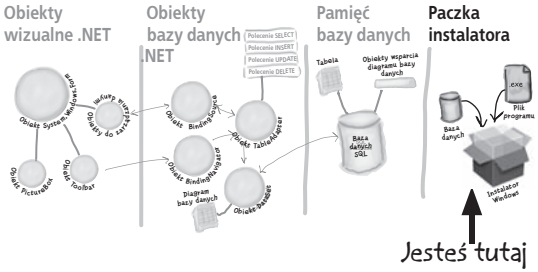
IDE najpierw buduje, potem wykonuje.

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 będą zapisywane — 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.

Budowanie programu nadpisuje dane w bazie.

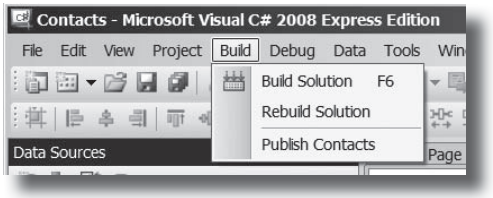
Jak zamienić TWOJĄ aplikację w aplikację WSZYSTKICH



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.

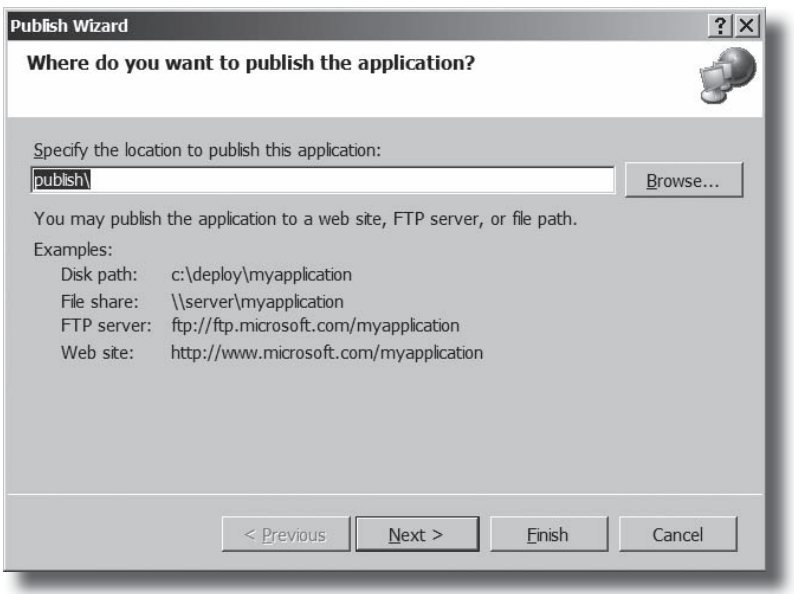
C# znacznie ułatwia przeprowadzenie procesu **wdrożenia** aplikacji. Wdrożenie aplikacji to zainstalowanie jej na innych komputerach. Z pomocą Visual C# IDE przygotowanie wdrożenia sprowadza się do dwóch etapów.

1 Wybierz *Publish Contacts* z menu Build.



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

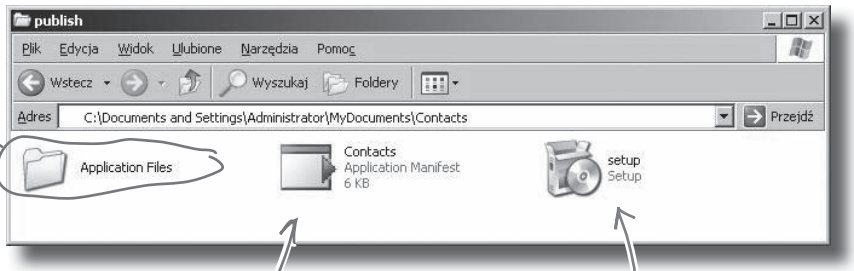
2 Zaakceptuj wszystkie domyślne wartości w oknie Publish Wizard poprzez naciśnięcie przycisku Finish. Zobaczysz, jak Twoja aplikacja będzie pakowana, a następnie ujrzysz folder, w którym znajdzie się plik setup.exe.



Daj innym użytkownikom możliwość korzystania z Twojej aplikacji

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

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



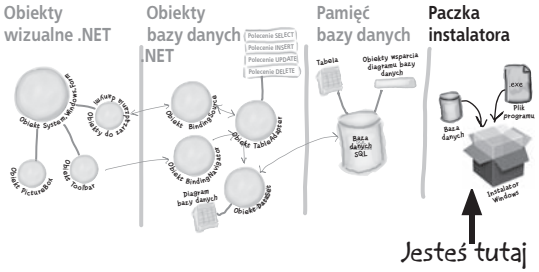
Ten plik udziela instalatorowi wszelkich informacji na temat tego, które pliki powinny być dołączone do programu podczas instalacji.

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! Pozostała tylko jedna rzecz do zrobienia, zanim pofruniesz szusować na stokach, chociaż...

Jeszcze nie skończyłeś: przetestuj instalację

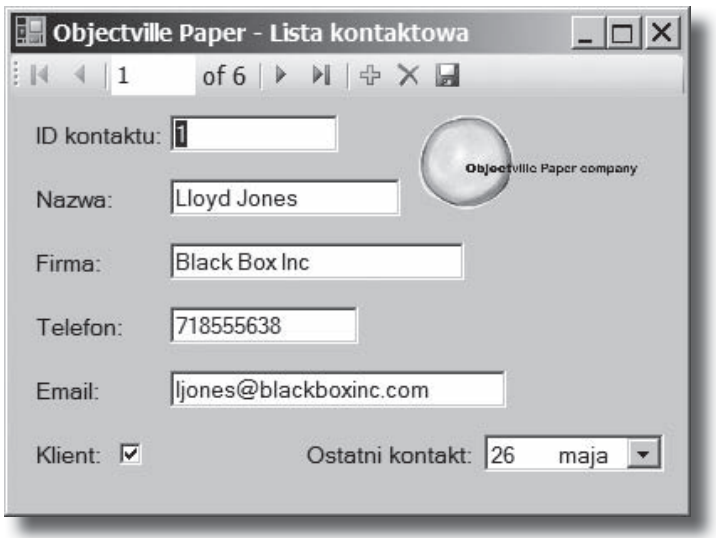


Zanim zaczniesz strzelać korkami od szampana, musisz przetestować wdrożenie i instalację. Nie dasz chyba nikomu swojego programu, dopóki sam go nie wypróbujesz?

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.

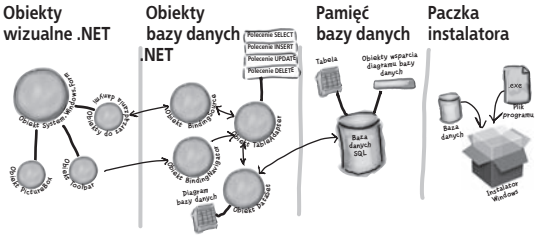
Każdy z sześciu rekordów, które na początku wpisałeś, będzie tutaj dostępny.

TESTUJ WSZYSTKO!

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

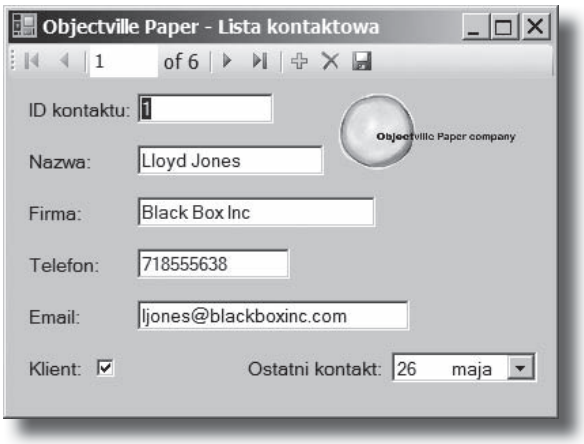
Stworzyłeś pełnowartościową aplikację bazodanową

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.