

» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

Head First C#. Edycja polska

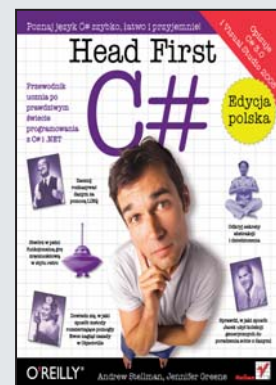
Autor: Andrew Stellman, Jennifer Greene

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!

Spis treści (streszczenie)

Wstęp	29
1 Zwiększ wydajność przy pomocy C#: <i>Aplikacje Visual Studio w 10 minut lub mniej</i>	41
2 To tylko kod: <i>Pod maską</i>	81
3 Obiekty: zorientuj się! <i>Tworzenie kodu ma sens</i>	121
4 Typy i referencje: <i>Jest 10:00. Czy wiesz, gdzie są Twoje dane?</i>	157
Laboratorium C# numer 1: <i>Dzień na wyścigach</i>	195
5 Ukrywanie: <i>Co ma być ukryte... niech będzie ukryte</i>	205
6 Dziedziczenie: <i>Drzewo genealogiczne Twoich obiektów</i>	237
7 Interfejsy i klasy abstrakcyjne: <i>Klasy, które dotrzymują swoich obietnic</i>	281
8 Typy wyliczeniowe i kolekcje: <i>Przechowywanie dużej ilości danych</i>	337
Laboratorium C# numer 2: <i>Wyprawa</i>	389
9 Odczyt i zapis plików: <i>Zapisz tablice bajtów, zapisz świat</i>	411
10 Obsługa wyjątków: <i>Gaszenie pożarów nie jest już popularne</i>	463
11 Zdarzenia i delegaty: <i>Co robi Twój kod, kiedy nie patrzysz</i>	505
12 Powtórka i pokaz: <i>Wiedza, moc i tworzenie ciekawych rzeczy</i>	537
13 Kontrolki i grafika: <i>Upiększ to</i>	585
14 Kapitan Wspaniały: <i>Śmierć obiektu</i>	643
15 LINQ: <i>Przejmij kontrolę nad danymi</i>	675
Laboratorium C# numer 3: <i>Invaders</i>	703
Dodatek A <i>Pozostałości</i>	725
Skorowidz	739

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#?

Dla kogo jest ta książka?	30
Wiemy, o czym myślisz	31
Metapoznanie: myślenie o myśleniu	33
Oto, co możesz zrobić, aby wysłać mózg na misję	35
Przeczytaj to	37
Grupa korektorów technicznych	38
Podziękowania	39

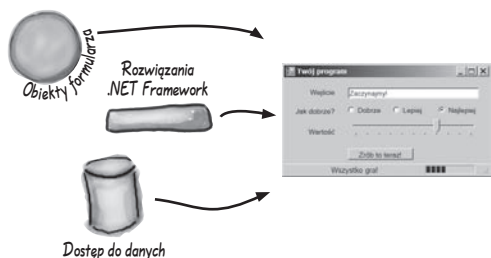
Zwiększ wydajność przy pomocy C#

1

Aplikacje Visual Studio w 10 minut lub mniej

Czy chcesz tworzyć wspaniałe programy naprawdę szybko?

Wraz z C# dostajesz do ręki **potężny język programowania** i wartościowe narzędzie. Dzięki **Visual Studio IDE** do historii przejdą sytuacje, w których musiałeś pisać jakiś nędzny kod, 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#	42
C# oraz Visual Studio ułatwiają wiele czynności	43
Pomóż dyrektorowi naczelnemu zrezygnować z papieru	44
Sprawdź potrzeby Twoich użytkowników, zanim zaczniesz tworzyć program	45
Oto program, który zamierzasz stworzyć	46
Co robisz w Visual Studio	48
Co Visual Studio robi za Ciebie	48
Stwórz interfejs użytkownika	52
Visual Studio za kulisami	54
Dodaj coś do kodu generowanego automatycznie	55
Możesz już uruchomić aplikację	56
Potrzebujemy bazy danych do przechowywania naszych informacji	58
Tworzenie tabeli dla listy kontaktowej	60
Pola na karcie kontaktowej stają się kolumnami w tabeli People	62
Zakończ tworzenie tabeli	65
Utwórz diagram dla swoich danych, aby aplikacja miała do nich dostęp	66
Wstaw dane z kart do bazy	68
Połącz formularz z bazą danych, korzystając ze źródeł danych	70
Dodaj kontrolki powiązane z bazą danych do formularza	72
Dobre programy są intuicyjne w użyciu	74
Jak zamienić TWOJĄ aplikację w aplikację WSZYSTKICH	77
Daj innym użytkownikom możliwość korzystania z Twojej aplikacji	78
Jeszcze nie skończyłeś: przetestuj instalację	79
Stworzyłeś pełnowartościową aplikację bazodanową	80

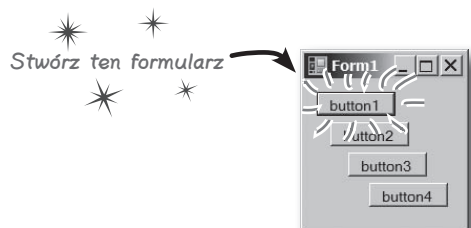
To tylko kod

2

Pod maską

Jesteś programistą, nie tylko użytkownikiem IDE.

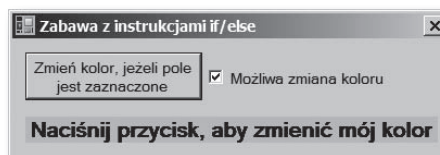
IDE może wykonać za Ciebie wiele pracy, ale na razie jest to wszystko, co może dla Ciebie zrobić. Oczywiście, istnieje wiele **powtarzalnych czynności** podczas pisania aplikacji i IDE okazuje się tu bardzo pomocne. Praca z nim to jednak *dopiero początek*. Możesz wycisnąć z swoich programów znacznie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



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

Klasa zawiera **kawałek** kodu Twojego programu (choć 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.



Kiedy robisz to...	82
... IDE robi to	83
Skąd się biorą programy	84
IDE pomaga Ci kodować	86
Kiedy zmieniasz coś w IDE, zmieniasz także swój kod	88
Anatomia programu	90
Twój program wie, skąd zacząć	92
Możesz zmienić punkt wejścia programu	94
W tej samej przestrzeni nazw mogą być dwie klasy	99
Twoje programy używają zmiennych do pracy z danymi	100
C# używa znanych symboli matematycznych	102
Pętle wykonują czynność wielokrotnie	103
Kodowanie czas zacząć	104
Instrukcje if/else podejmują decyzje	105
Ustal warunki i sprawdź, czy są prawdziwe	106

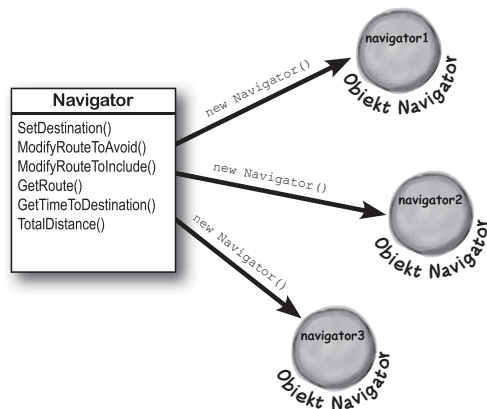
Obiekty: zorientuj się!

3

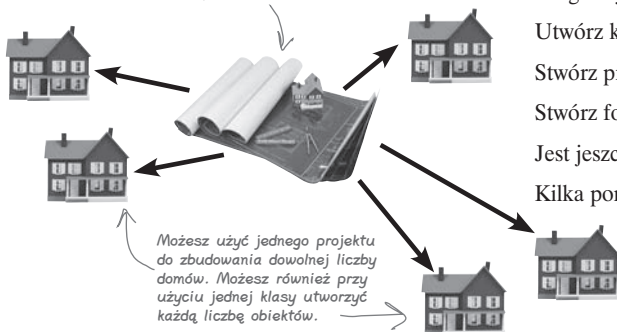
Tworzenie kodu ma sens

Każdy pisany przez Ciebie program rozwiązuje jakiś problem.

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 przeznaczony 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.



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



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

W jaki sposób Mike myśli o swoich problemach	122
W jaki sposób system nawigacyjny w samochodzie Mike'a rozwiązuje jego problemy	123
Klasa Navigator napisana przez Mike'a posiada metody do ustalania i modyfikacji tras	124
Wykorzystaj to, czego się nauczyłeś, do napisania prostej aplikacji	125
Mike ma pewien pomysł	126
Mike może użyć obiektów do rozwiązania swojego problemu	127
Używasz klasy do utworzenia obiektu	128
Kiedy tworzysz obiekt na podstawie klasy, to taki obiekt nazywamy instancją klasy	129
Lepsze rozwiązanie... uzyskane dzięki obiektom!	130
Instancja używa pól do przechowywania danych na temat różnych rzeczy	134
Stwórzmy kilka instancji!	135
Dzięki za pamięć	136
Co Twój program ma na myśli	137
Możesz używać nazw klas i metod w celu uczynienia kodu bardziej intuicyjnym	138
Nadaj swojej klasie naturalną strukturę	140
Diagramy klas pozwalają w sensowny sposób zorganizować klasy	142
Utwórz klasę do pracy z kilkoma facetami	146
Stwórz projekt dla facetów	147
Stwórz formularz do interakcji z facetami	148
Jest jeszcze prostszy sposób inicjalizacji obiektów	151
Kilka pomysłów na projektowanie intuicyjnych klas	152

Typy i referencje

4

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

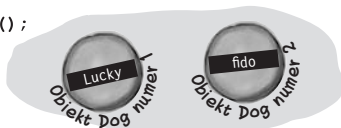
Bez danych Twoje programy są bezużyteczne. Potrzebujesz od użytkowników informacji. Na jej podstawie wyszukujesz lub tworzysz nową **informację** i zwracasz ją użytkownikom. W rzeczywistości prawie wszystko, co robisz podczas programowania, sprowadza się do **pracy z danymi** w taki czy w inny sposób. 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** (*psst... obiekty to także dane*).

Typ zmiennej określa rodzaj danych, jakie zmienna może przechowywać	158
Zmienna jest jak kubek z danymi	160
10 kilogramów danych w pięciokilogramowej torebce	161
Nawet wtedy, gdy liczba ma prawidłowy rozmiar, nie możesz przypisać jej do każdej zmiennej	162
Kiedy rzutujesz wartość, która jest zbyt duża, C# dopasuje ją automatycznie	163
C# przeprowadza niektóre rzutowania automatycznie	164
Kiedy wywołujesz metodę, zmienne muszą pasować do typów parametrów	165
Połączenie = z operatorem	170
Obiekty także są zmiennymi	171
Korzystaj ze swoich obiektów przy pomocy zmiennych referencyjnych	172
Referencje są jak etykiety do Twoich obiektów	173
Jeżeli nie ma już żadnej referencji, Twoje obiekty są usuwane z pamięci	174
Referencje wielokrotne i ich efekty uboczne	175
Dwie referencje oznaczają DWA sposoby na zmianę danych obiektu	180
Specjalny przypadek: tablice	181
Tablice mogą także zawierać grupę zmiennych referencyjnych	182
Witamy w barze Niechlujny Janek — najtańsze kanapki w mieście!	183
Obiekty używają referencji do komunikacji między sobą	185
Tam gdzie obiektów jeszcze nie było	186

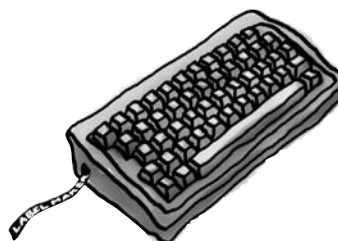
```
Dog fido;
Dog lucky = new Dog();
```



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



```
lucky = null;
```



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.

Specyfikacja: stwórz symulator wyścigów

196

Końcowy produkt

204



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**.



Agent CIA

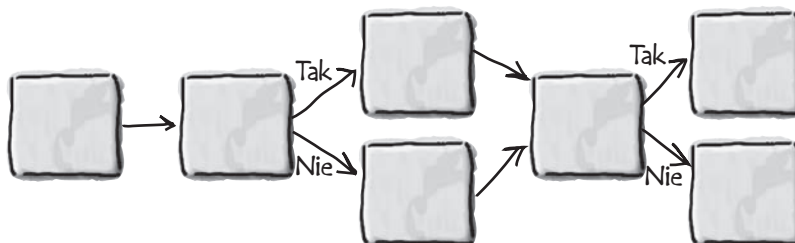


Agent KGB



Agent MI5

Krystyna planuje przyjęcia	206
Co powinien robić program szacujący?	207
Jazda próbna Krystyny	212
Każda opcja powinna być obliczana indywidualnie	214
Bardzo łatwo przez przypadek źle skorzystać z obiektów	216
Hermetyzacja oznacza, że niektóre dane w klasie są prywatne	217
Użyj hermetyzacji w celu kontroli dostępu do metod i pól Twojej klasy	218
Ale czy jego prawdziwa tożsamość jest NAPRAWDĘ chroniona?	219
Dostęp do prywatnych pól i metod można uzyskać tylko z wnętrza klasy	220
Kilka sugestii dotyczących hermetyzacji	223
Hermetyzacja utrzymuje Twoje dane w nieskazitelnym stanie	224
Właściwości sprawiają, że hermetyzacja będzie łatwiejsza	225
Stwórz aplikację do przetestowania klasy Farmer	226
Użyj automatycznych właściwości do ukończenia klasy	227
Co wtedy, gdy chcemy zmienić pole mnożnika wyżywienia?	228
Użyj konstruktora do inicjalizacji pól prywatnych	229



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**.



Krystyna organizuje także przyjęcia urodzinowe	238
Potrzebujemy klasy BirthdayParty	239
Jeszcze jedna rzecz... Czy możesz dodać opłatę 100 zł za przyjęcia powyżej 12 osób?	245
Kiedy klasy używają dziedziczenia, kod musi być napisany tylko raz	246
Zbuduj model klasy, rozpoczynając od rzeczy ogólnych i przechodząc do bardziej konkretnych	247
W jaki sposób zaprojektowałbyś symulator zoo?	248
Użyj dziedziczenia w celu uniknięcia zwielokrotniania kodu w klasach potomnych	249
Różne zwierzęta wydają różne dźwięki	250
Pomyśl, w jaki sposób pogrupować zwierzęta	251
Stwórz hierarchię klas	252
Każda klasa pochodna rozszerza klasę bazową	253
Aby dziedziczyć z klasy bazowej, użyj dwukropka	254
Wiemy, że dziedziczenie dodaje pola, właściwości i metody klasy bazowej...	257
Klasa pochodna może przesłaniać odziedziczone metody w celu ich modyfikacji lub zmiany	258
W każdym miejscu, gdzie możesz użyć klasy bazowej, możesz zamiast tego użyć jednej z jej klas pochodnych	259
Klasa potomna może uzyskać dostęp do klasy bazowej, używając słowa kluczowego base	264
Jeśli Twoja klasa bazowa posiada konstruktor, klasa pochodna też musi go mieć	265
Teraz jesteś już gotowy do dokończenia zadania Krystyny	266
Stwórz system zarządzania ulem	271
Najpierw stworzysz system podstawowy	272
Użyj dziedziczenia, aby rozszerzyć system zarządzania pszczołami	276

Interfejsy i klasy abstrakcyjne

7

Klasy, które dotrzymują swoich obietnic

Czyny potrafią powiedzieć więcej niż słowa.

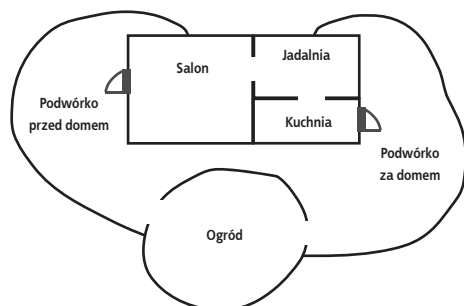
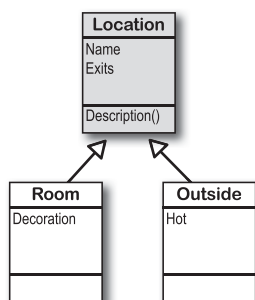
Czasami potrzebujesz pogrupować swoje obiekty na podstawie tego, **co robią**, zamiast tego, z jakiej klasy dziedziczą. To jest moment, w którym należy powiedzieć o **interfejsach**. Pozwalają one na pracę z każdą klasą, która jest w stanie wykonać daną czynność. Z **wielkimi możliwościami przychodzą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

* Dziedziczenie

* Abstrakcja

* Hermetryzacja

* Polimorfizm



Wróćmy do pszczelej korporacji	282
Możemy użyć dziedziczenia do utworzenia klas dla różnych typów pszczół	283
Interfejs daje klasie do zrozumienia, że musi zaimplementować określone metody i właściwości	284
Użyj słowa kluczowego interface do zdefiniowania interfejsu	285
Teraz możesz utworzyć instancję NectarStinger, która będzie wykonywała dwa rodzaje zadań	286
Klasy implementujące interfejsy muszą zawierać WSZYSTKIE ich metody	287
Poćwicz trochę z interfejsami	288
Nie możesz stworzyć instancji interfejsu, ale możesz uzyskać jego referencję	290
Referencje interfejsów działają tak samo jak referencje obiektów	291
Za pomocą „is” możesz sprawdzić, czy klasa implementuje określony interfejs	292
Interfejsy mogą dziedziczyć z innych interfejsów	293
RoboBee 4000 może wykonywać zadania pszczół bez potrzeby spożywania cennego miodu	294
is określa, co obiekt implementuje, a mówi kompilatorowi, jak go traktować	295
Ekspres do kawy także jest urządzeniem	296
Rzutowanie w górę działa w odniesieniu do obiektów i interfejsów	297
Rzutowanie w dół pozwala zamienić urządzenie z powrotem w ekspres do kawy	298
Rzutowanie w górę i w dół działa także w odniesieniu do interfejsów	299
Jest coś więcej niż tylko public i private	303
Modyfikatory dostępu zmieniają zasięg	304
Obiekty niektórych klas nigdy nie powinny być tworzone	307
Klasa abstrakcyjna jest jak skrzyżowanie klasy i interfejsu	308
Obiekty niektórych klas nigdy nie powinny być tworzone	310
Metoda abstrakcyjna nie ma ciała	311
Polimorfizm oznacza, że jeden obiekt może przyjmować wiele różnych postaci	319

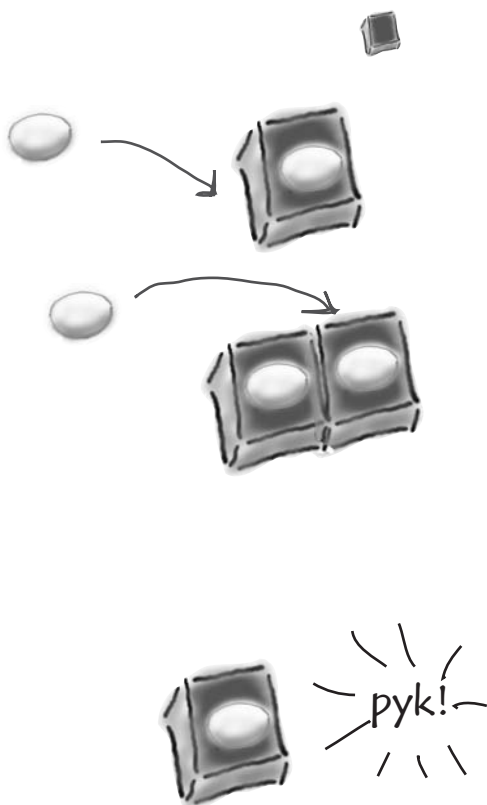
Typy wyliczeniowe i kolekcje

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.



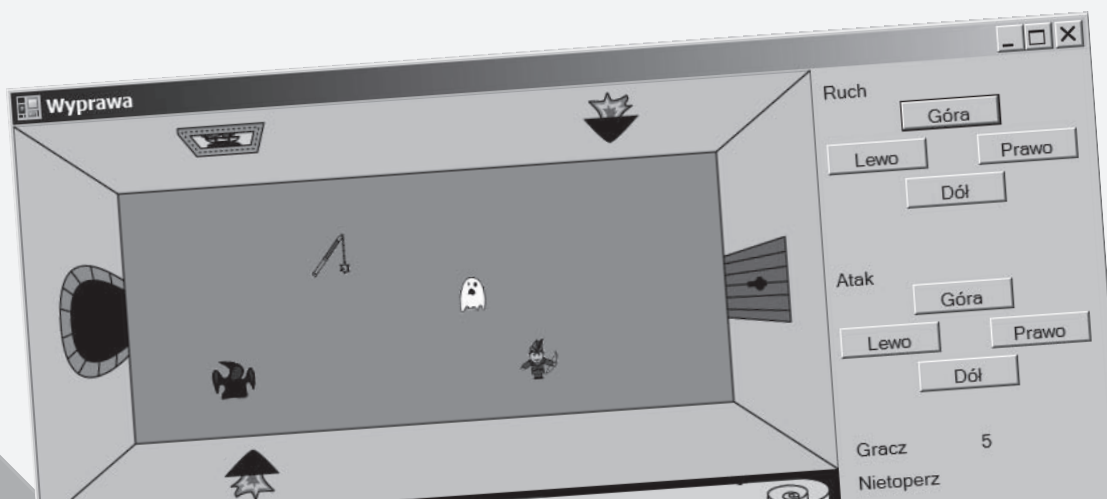
Łańcuchy znaków nie zawsze sprawdzają się przy przechowywaniu kategorii danych	338
Typy wyliczeniowe pozwalają Ci wyliczyć prawidłowe wartości	339
Typy wyliczeniowe pozwalają na reprezentowanie liczb za pomocą nazw	340
Możesz użyć tablicy, aby stworzyć talię kart...	343
Z tablicami ciężko się pracuje	344
Listy ułatwiają przechowywanie kolekcji... czegokolwiek	345
Listy są bardziej elastyczne niż tablice	346
Listy kurczą się i rosną dynamicznie	349
Obiekty List mogą przechowywać każdy typ	350
Inicjalizatory kolekcji działają tak samo jak inicjalizatory obiektu	354
Stwórzmy listę kaczek	355
Listy są proste, ale SORTOWANIE może być skomplikowane	356
Dwa sposoby na posortowanie kaczek	357
Użyj interfejsu IComparer, aby powiedzieć liście, jak ma sortować	358
Stwórz instancję obiektu porównującego	359
IComparer może wykonywać złożone porównania	360
Użyj słownika do przechowywania kluczy i wartości	363
Ograniczenia funkcjonalności słownika	364
Twoje klucze i wartości mogą być także różnego typu	365
Możesz tworzyć własne przeciążone metody	371
I jeszcze WIĘCEJ typów kolekcji...	383
Kolejka działa według reguły: pierwszy przyszedł, pierwszy wyszedł	384
Stos działa według reguły: ostatni przyszedł, pierwszy wyszedł	385

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.

Specyfikacja: stwórz grę przygodową 390

Zabawa dopiero się zaczyna! 410



Odczyt i zapis plików

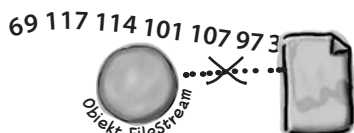
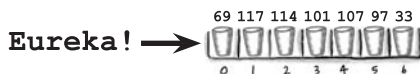
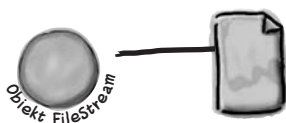
9

Zapisz tablice bajtów, zapisz świat

Czasami oplaća się być trwałym.

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

C# używa strumieni do zapisu i odczytu danych	412
Różne strumienie zapisują i odczytują różne rzeczy	413
FileStream zapisuje bajty do pliku	414
Zapis i odczyt wymaga dwóch obiektów	419
Dane mogą przechodzić przez więcej niż jeden strumień	420
Użyj wbudowanych obiektów do wyświetlenia standardowych okien dialogowych	423
Okna dialogowe także są obiektami	425
Używaj wbudowanych klas File oraz Directory do pracy z plikami i katalogami	426
Używaj okien dialogowych do otwierania i zapisywania plików	429
Dzięki IDisposable obiekty usuwane są prawidłowo	431
Unikaj błędów systemowych, korzystając z instrukcji using	432
Zapisywanie danych do plików wymaga wielu decyzji	438
Użyj instrukcji switch do wyboru właściwej opcji	439
Dodaj przeciążony konstruktor Deck(), który wczytuje karty z pliku	441
Co dzieje się z obiektem podczas serializacji?	443
Czym w istocie JEST stan obiektu? Co musi zostać w nim zapisane?	444
Kiedy obiekt jest serializowany, serializowane są także wszystkie obiekty z nim powiązane...	445
Serializacja pozwala Ci zapisywać lub odczytywać całe obiekty na raz	446
Jeżeli chcesz stosować serializację w odniesieniu do klasy, to musisz oznaczyć ją atrybutem [Serializable]	447
.NET automatycznie konwertuje tekst do postaci Unicode	451
C# może użyć tablicy bajtów do przesyłania danych	452
Do zapisywania danych binarnych używaj klasy BinaryWriter	453
Pliki utworzone dzięki serializacji mogą być czytane także ręcznie	455
StreamReader i StreamWriter będą do tego odpowiednie	459



Obsługa wyjątków

10

Gaszenie pożarów nie jest już popularne**Programiści nie mają być strażakami.**

Pracowałeś jak wół, przebrnąłeś przez dokumentację techniczną 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**.

Damian potrzebuje swoich wymówek, aby być mobilnym	464
Kiedy program wyrzuca wyjątek, .NET tworzy obiekt Exception	468
Kod Damiana zrobił coś nieoczekiwane	470
Wszystkie obiekty wyjątków dziedziczą z Exception	472
Debugger pozwala Ci wyśledzić wyjątki w kodzie i zapobiec im	473
Użyj debugera wbudowanego w IDE, aby znaleźć problem w programie do zarządzania wymówkami	474
Oj, oj! — w kodzie dalej są błędy...	477
Obsłuż wyjątki za pomocą try i catch	479
Co się stanie, jeżeli wywołwana metoda jest niebezpieczna?	480
Użyj debugera do prześledzenia przepływu try/catch	482
Jeśli posiadasz kod, który powinien być uruchomiony ZAWSZE, zastosuj finally	484
Użyj obiektu Exception w celu uzyskania informacji o problemie	489
Użyj więcej niż jednego bloku catch do wyłapania różnych typów wyjątków	490
Jedna klasa wyrzuca wyjątek, inna klasa go wyłapuje	491
Pszczoly i ich wyjątek OutOfHoney	492
Łatwy sposób na uniknięcie licznych problemów: using umożliwia Ci stosowanie try i finally za darmo	495
Unikanie wyjątków: zaimplementuj IDisposable, aby przeprowadzić własne procedury sprzątnięcia	496
Najgorszy z możliwych blok catch: komentarze	498
Tymczasowe rozwiązania są dobre (tymczasowo)	499
Kilka wskazówek dotyczących obsługi wyjątków	500
Damian w końcu pojechał na urlop...	503



Zdarzenia i delegaty

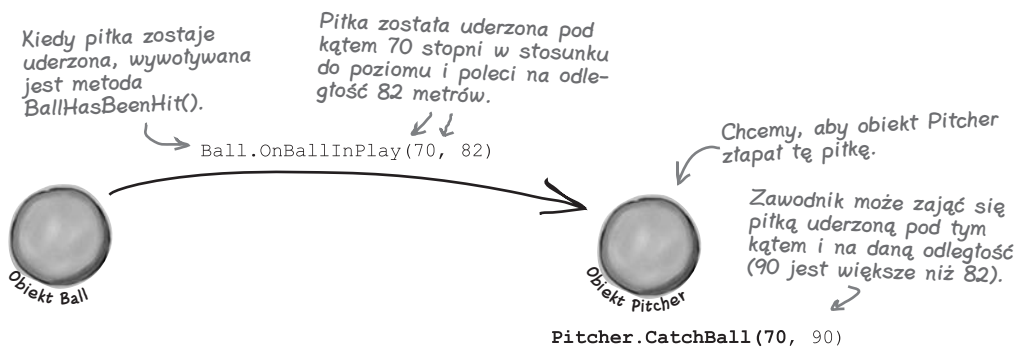
11

Co robi Twój kod, kiedy nie patrzysz

Twoje obiekty zaczynają myśleć o sobie.

Nie możesz zawsze kontrolować tego, co robią Twoje obiekty. Czasami różne rzeczy... zdarzają się. Kiedy to następuje, chciałbyś, aby Twoje obiekty były wystarczająco sprytnie i odpowiednio **reagowały**. To miejsce, w którym do akcji wkraczają zdarzenia. Jeden obiekt *udostępnia* zdarzenie, inny je *obsługuje* i wszystko pracuje razem, aby całość działała sprawnie. Jest to wspaniałe, o ile nie masz zbyt wielu obiektów podpiętych pod to samo zdarzenie. Wtedy bardzo pomocne okazują się **funkcje zwrotne**.

Czy kiedykolwiek marzyłeś o tym, aby Twoje obiekty potrafiły samodzielnie myśleć?	506
Ale skąd obiekt WIE, że ma odpowiedzieć?	506
Kiedy wystąpi ZDARZENIE... obiekty nasłuchują	507
Jeden obiekt wywołuje zdarzenie, inne nasłuchują...	508
Potem inne obiekty obsługują zdarzenie	509
Łącząc punkty	510
IDE automatycznie tworzy za Ciebie funkcje obsługi zdarzeń	514
Wszystkie formularze, które utworzyłeś, używają zdarzeń	520
Połączenie nadawców zdarzenia z jego odbiorcami	522
Delegat ZASTĘPUJE właściwą metodę	523
Delegat w akcji	524
Każdy obiekt może subskrybować publiczne zdarzenie...	527
Użyj zamiast zdarzenia funkcji zwrotnej, aby podpiąć dokładnie jeden obiekt do delegatu	529
Funkcje zwrotne używają delegatu, NIE zdarzeń	530



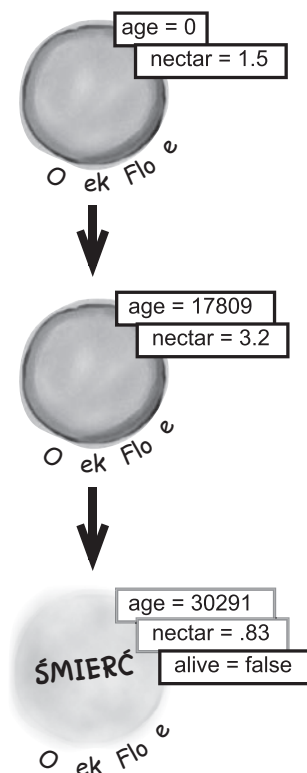
Powtórka i pokaz

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

Przebyłeś długą drogę	538
Zajmowaliśmy się także pszczołami	539
Architektura symulatora ula	540
Budowanie symulatora ula	541
Życie i śmierć kwiatów	545
Potrzebujemy teraz klasy Bee	546
Wypełnianie klasy Hive	554
Metoda Go() klasy Hive	555
Jesteśmy gotowi na stworzenie świata	556
Tworzymy system turowy	557
Uczenie pszczół zachowań	564
Główny formularz wywołuje Go() dla całego świata	566
Możemy użyć obiektu World do pobrania statystyk	567
Zegary sygnalizują zdarzenia wielokrotnie	568
Zegar w tle używa delegata	569
Pracujemy z grupami pszczół	576
Kolekcje kolekcjonują... DANE	577
LINQ ułatwia pracę z danymi w kolekcjach i bazach danych	579

Kontrolki i grafika

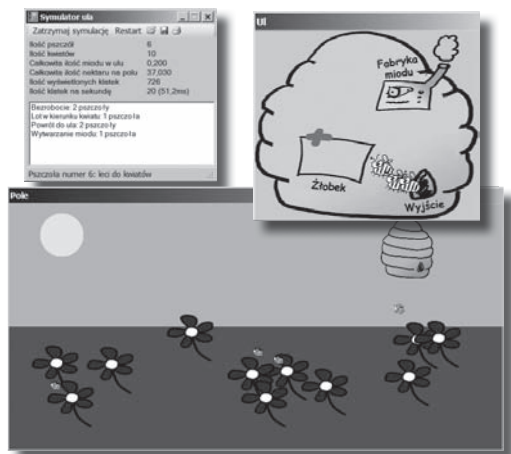
13

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.

Cały czas do interakcji z programami używałeś kontroltek	586
Kontrolki formularza są tylko obiektami	587
Dodaj do projektu rendering	590
Kontrolki są dobrze przystosowane do wyświetlania różnych elementów wizualnych	592
Stwórz swoją pierwszą animowaną kontrolkę	595
Twoje kontrolki także muszą usuwać swoje kontrolki!	599
UserControl to dobry sposób na tworzenie kontroltek	600
Dodaj do projektu formularze reprezentujące ul i pole	604
Stwórz klasę Renderer	605
Przyjrzyjmy się bliżej sprawom wydajności	612
Zmieniłeś rozmiar bitmap przy pomocy obiektu Graphics	614
Zasoby Twoich obrazków przechowywane są w postaci obiektów Bitmap	615
Użyj System.Drawing do PRZEJĘCIA KONTROLI nad grafiką	616
30-sekundowa podróż do świata tajemnic grafiki GDI+	617
Użyj Graphics, aby na formularzu narysować obrazek	618
Klasa Graphics może usunąć problem przezroczystości...	623
Użyj zdarzenia Paint, aby grafika była mocno związana z formularzem	624
Bliższe spojrzenie na sposób rysowania formularzy i kontroltek	627
Podwójne buforowanie czyni animację bardziej płynną	630
Podwójne buforowanie jest wbudowane w formularze i kontrolki	631
Użyj obiektu Graphics i procedury obsługi zdarzenia do drukowania	636
PrintDocument pracuje z obiektem okna dialogowego drukowania i obiektem okna podglądu wydruku	637

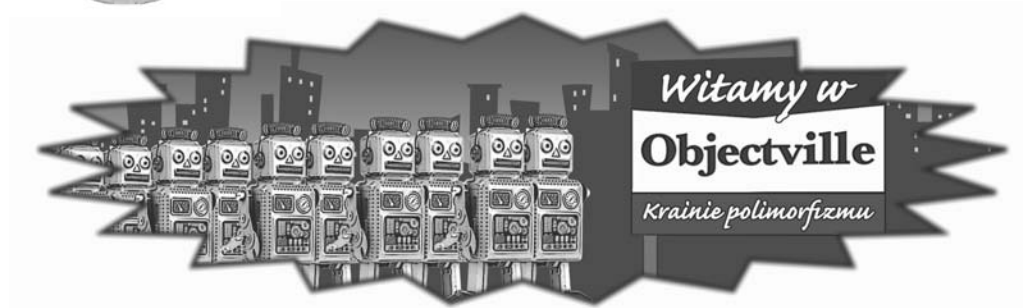


14

Kapitan Wspaniały

Śmierć obiektu

Kapitan Wspaniały, najbardziej zdumiewający obiekt Objectville, próbuje pokonać arcyźródło zła...	644
Twoją ostatnią szansą na ZROBIENIE czegoś... jest użycie finalizatora	650
Kiedy DOKŁADNIE wywoływany jest finalizator?	651
Dispose() działa z using, a finalizatory działają z mechanizmem odzyskiwania elementów bezużytecznych	652
Finalizatory nie mogą polegać na stabilności	654
Spraw, aby obiekt serializował się w Dispose()	655
W międzyczasie na ulicach Objectville...	658
Struktura jest <i>podobna</i> do obiektu...	659
... ale <i>nie jest</i> na stercie	659
Wartości są kopiowane, referencje są przypisywane	660
Struktury traktowane są jak typy wartościowe, obiekty jak typy referencyjne	661
Stos i sarta: więcej na temat pamięci	663
Kapitan Wspaniały... nie tak bardzo	667
Metody rozszerzające zwiększają funkcjonalność ISTNIEJĄCYCH klas	668
Rozszerzanie podstawowego typu: string	670



LINQ

15

Przejmij kontrolę nad danymi

To świat przepętniony 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**.

Łatwy projekt...	676
...ale dane są w różnych miejscach	677
Dzięki LINQ możesz pobrać dane z różnych źródeł	678
Kolekcje .NET są przystosowane do działania z LINQ	679
LINQ ułatwia wykonywanie zapytań	680
LINQ jest prosty, ale Twoje zapytania wcale takie być nie muszą	681
LINQ ma wiele zastosowań	684
LINQ może połączyć Twoje wyniki w grupy	689
Połącz wartości Janka w grupy	690
Użyj join do połączenia dwóch kolekcji w jednym zapytaniu	693
Janek zaoszczędził kupę szmalu	694
Połącz LINQ z bazą danych SQL	696
Użyj join, aby połączyć Starbuzz i Objectville	700



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.

Dziadek wszystkich gier	704
Można zrobić znacznie więcej...	723



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.

1. Zastosowanie LINQ do XML	726
2. Refaktoryzacja	728
3. Niektóre z naszych ulubionych komponentów okna Toolbox	730
4. Aplikacje konsolowe	732
5. Windows Presentation Foundation	734
Czy wiesz, że C# i .NET Framework potrafią...	736

S

Skorowidz

739

backgroundWorker 1

fileSystemWatcher 1

performanceCounter 1



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;
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());
        }
    }
}
```

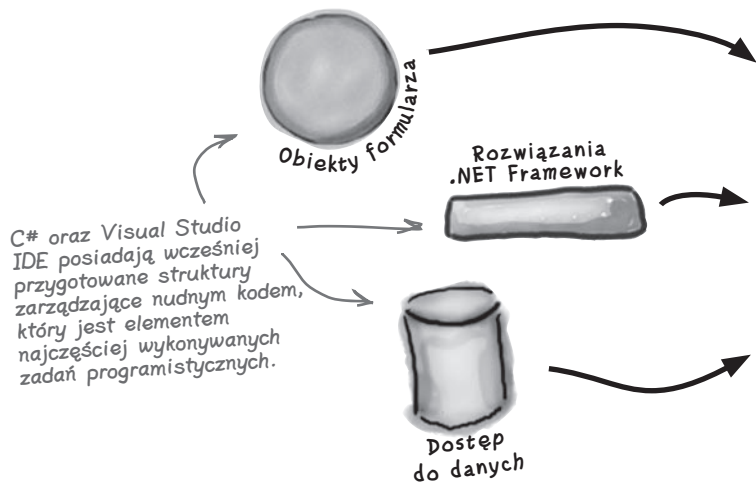
```
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(105, 56);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(75, 23);
    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

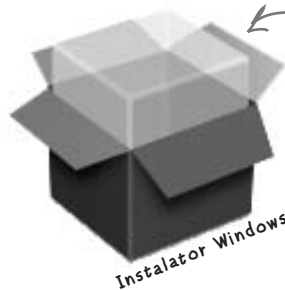
Firma papiernicza Objectville Paper właśnie zatrudniła nowego dyrektora naczelnego. Uwielbia on wycieczki piesze, kawę, przyrodę... i podjął decyzję, że będzie przyczyniał się do ratowania lasów. Ma potrzebę zostania kierownikiem „elektronicznym”. Na pierwszy ogień pójdzie lista jego kontaktów. W najbliższy 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 znajdują się na każdym komputerze.



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

- 2 Dział sprzedaży firmy papierniczej 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.

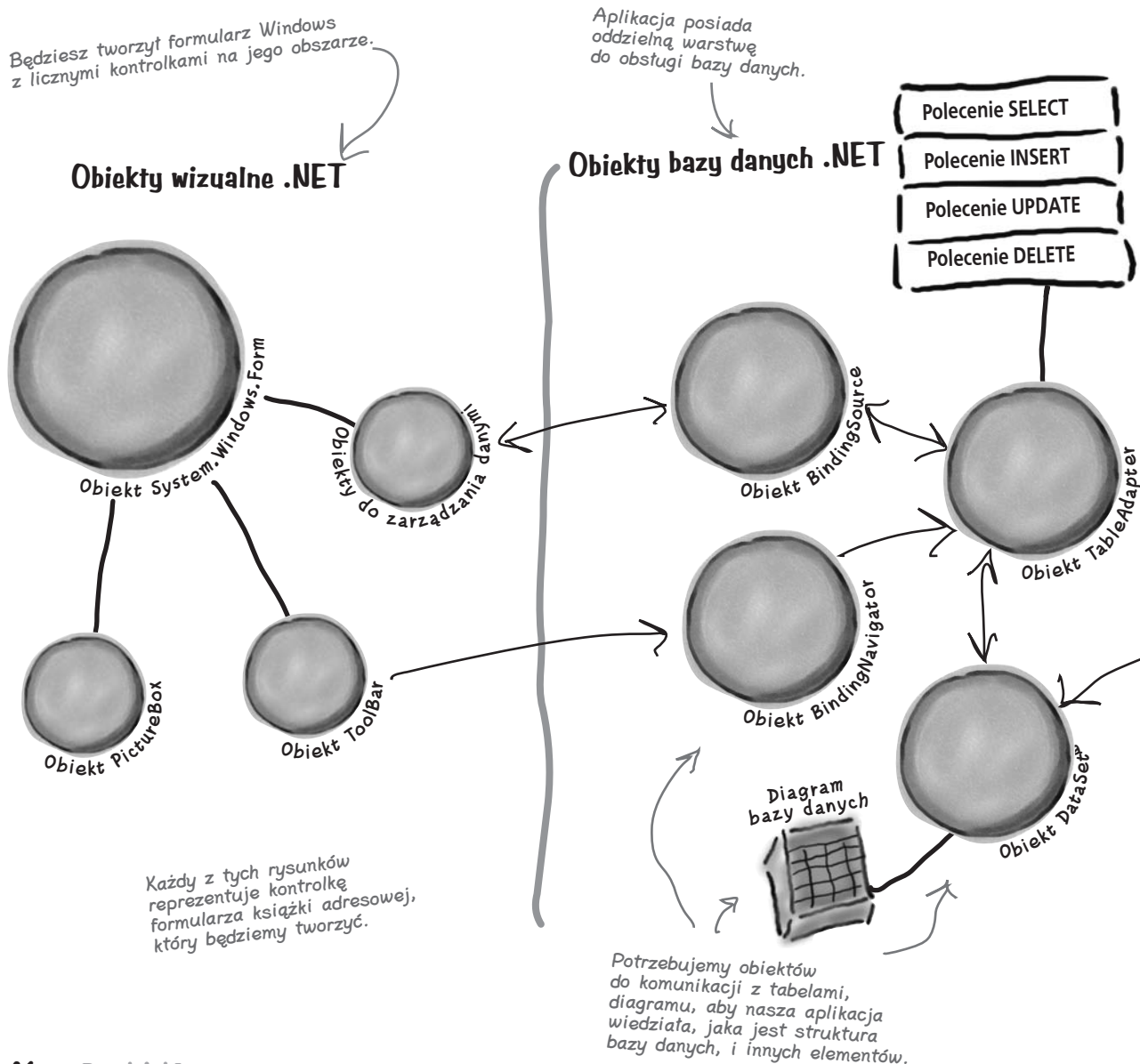
Wiemy już, że Visual Studio ułatwia pracę z bazami danych. Przechowywanie informacji w takiej formie pozwala na jednoczesny dostęp do nich dyrektorowi naczelnemu oraz działowi sprzedaży, mimo że fizycznie istnieje tylko jedna kopia danych.



Oto program, który zamierzasz stworzyć

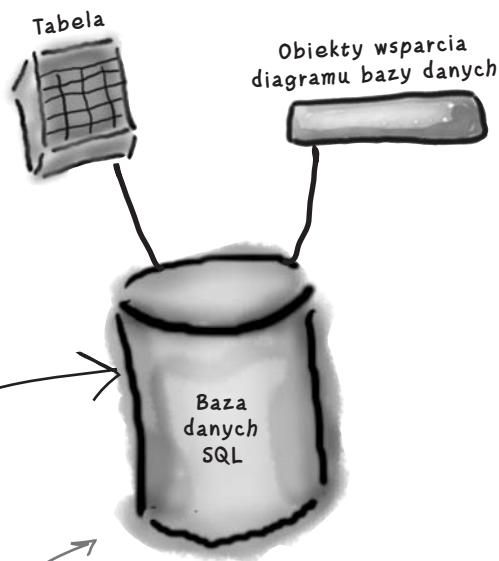
Potrzebujesz aplikacji z graficznym interfejsem użytkownika, obiektów niezbędnych do komunikacji z bazą danych, właściwej bazy danych oraz programu instalacyjnego. Brzmi to jak wyrok i zapowiada konieczność wygosparowania ogromnej ilości czasu, ale zbudujesz to wszystko po przeanalizowaniu następujących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

A oto struktura programu, który zamierzamy stworzyć:



Wszystkie dane przechowywane są w tabeli bazy danych SQL Express.

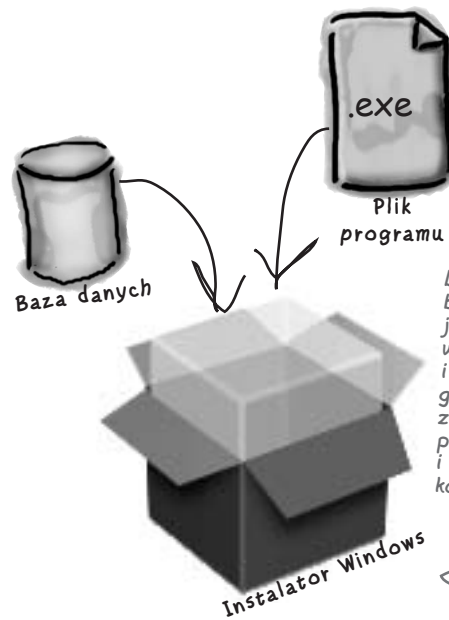
Pamięć bazy danych



To jest prawdziwa baza danych, którą możemy utworzyć i zarządzać przy pomocy Visual Studio.

Program po utworzeniu umieszczany jest w pliku instalatora Windows.

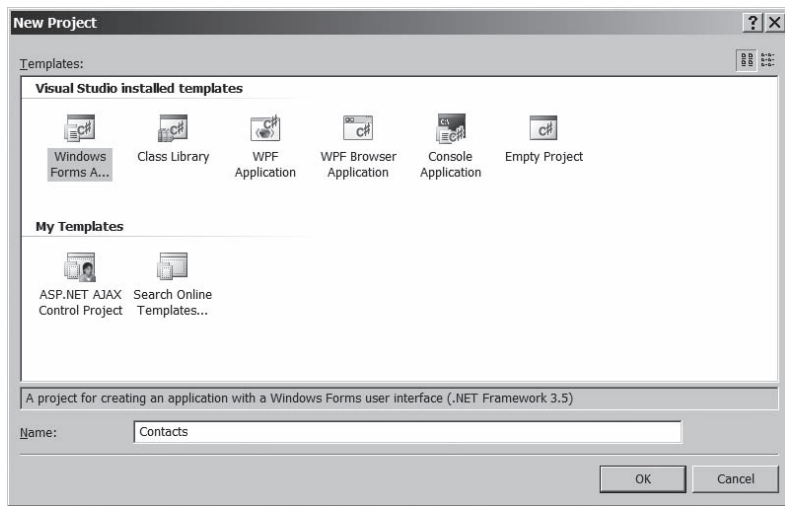
Paczka instalatora



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.



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 zapiszesz go, wybierając „Save All” z menu File — spowoduje to zapisanie do katalogu wszystkich jego plików. Jeżeli wybierzesz „Save”, zapisany zostanie tylko plik, nad którym aktualnie pracujesz.

Ten plik zawiera kod C# definiujący zachowanie się formularza.

Ten zawiera kod uruchamiający program i wyświetlający formularz.

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



Form1.cs



Program.cs



Form1.Designer.cs

Visual Studio tworzy wszystkie trzy pliki automatycznie.



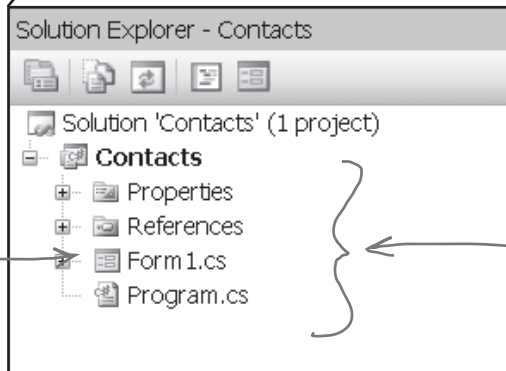
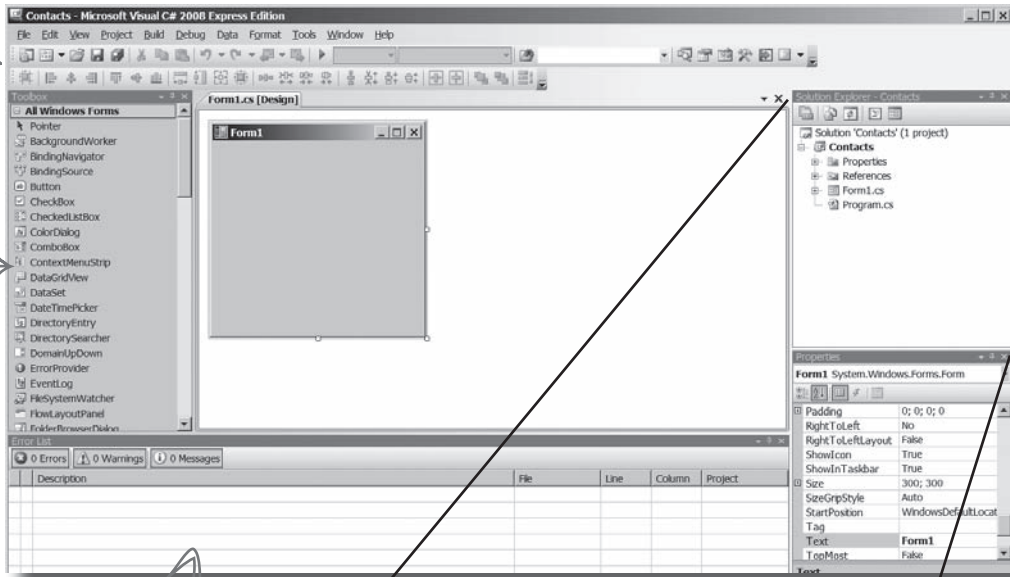
Zaostrz ołówek

Poniżej pokazany jest wygląd ekranu, jaki prawdopodobnie możesz w tej chwili zaobserwować. Powinieneś odgadnąć, do czego służą poszczególne okna, bazując na wiedzy, którą na tym etapie posiadasz. 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.



Poznaj swoje IDE



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.

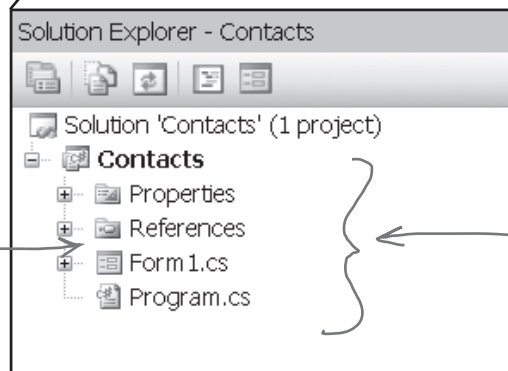
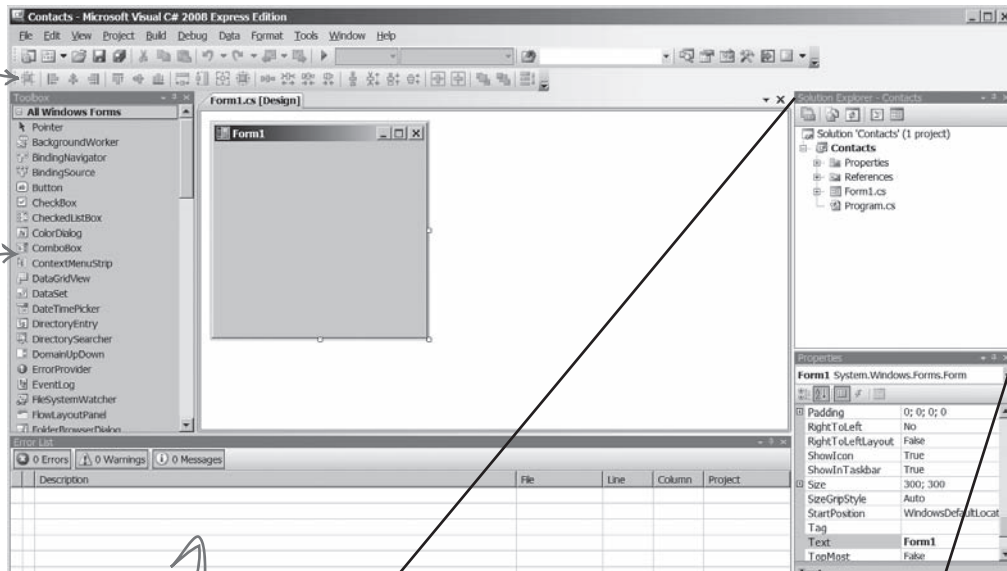
To jest okno z kontrolkami. Zawiera zestaw kontrolek, które możesz przeciągać wprost na formularz.

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.

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

Możesz przetaczać 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?

U: 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?

U: 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. Próbuując 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?

U: Możesz go zmienić. IDE jest przystosowane do tworzenia kodu na podstawie tego, w jaki sposób elementy zostały przeciągnięte lub dodane do aplikacji. Czasami nie jest to sposób, jaki byłby przez Ciebie 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?

U: 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?

U: 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?

U: 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.

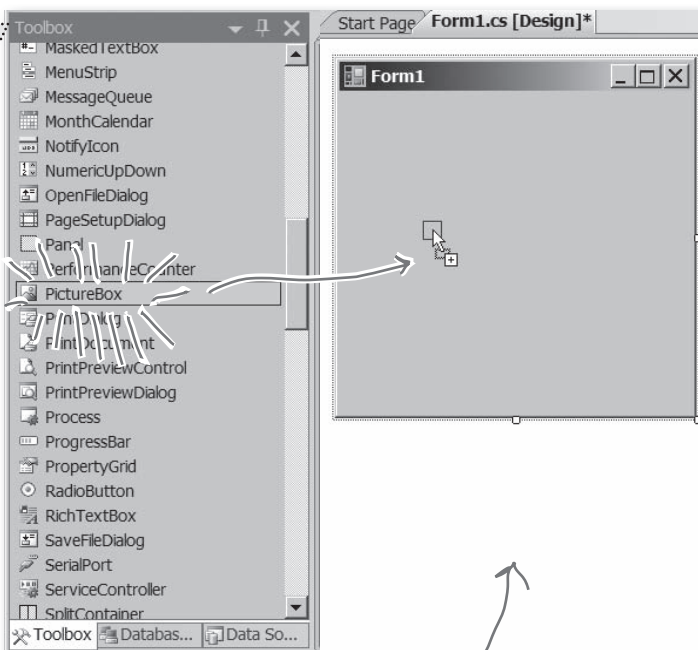
Stwórz interfejs użytkownika

Dodawanie kontrolki 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.

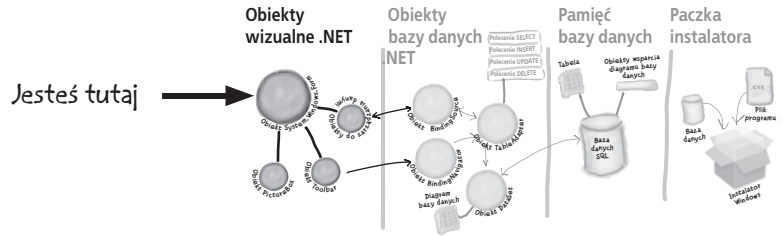


Form1.Designer.cs



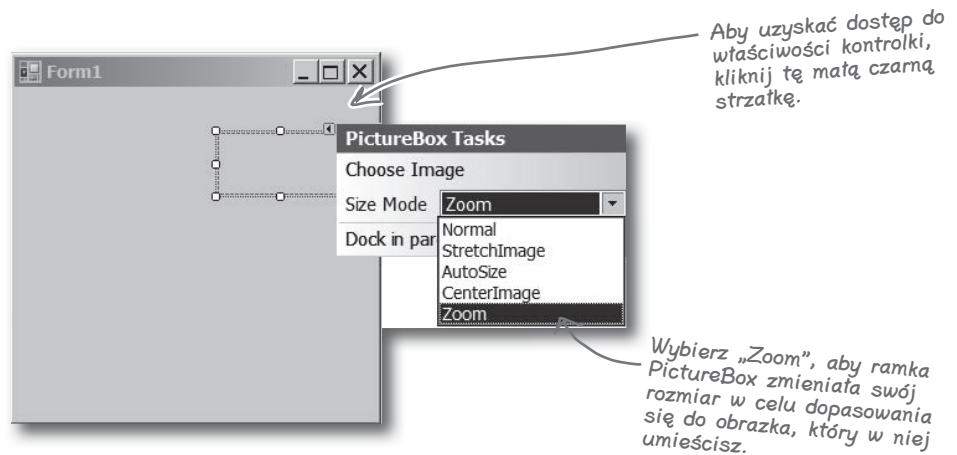
Nie przejmuj się, jeżeli nie jesteś profesjonalistą w projektowaniu graficznych interfejsów użytkownika.

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.



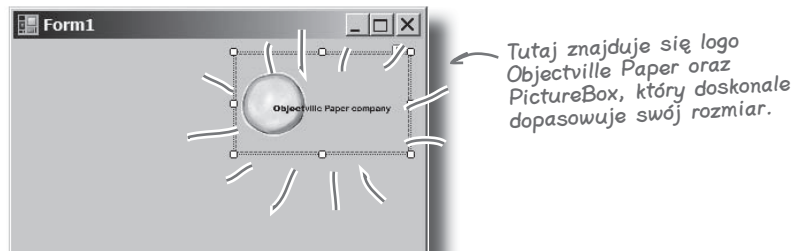
2 Ustaw tryb Zoom w PictureBox.

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.

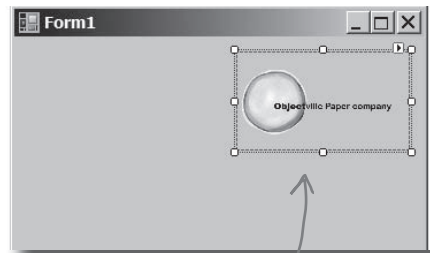
Pobierz logo Objectville Paper z serwera ftp (<ftp://ftp.helion.pl/przyklady/hfcsk.zip>) i zapisz je na twardym dysku. Następnie wybierz strzałkę właściwości i kliknij Choose Image. Zaznacz pole wyboru Local resource, a następnie kliknij Import i znajdź Logo. Wszystko powinno ustawić się poprawnie.



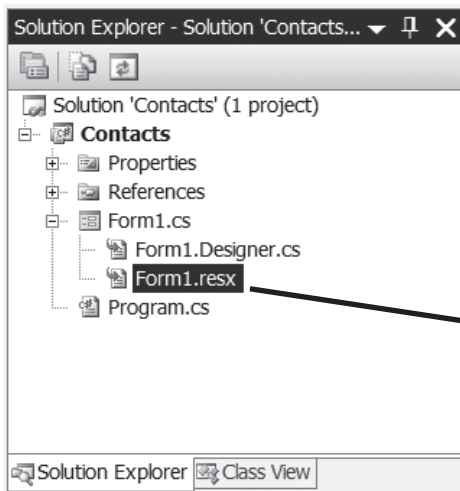
Visual Studio za kulisami

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.

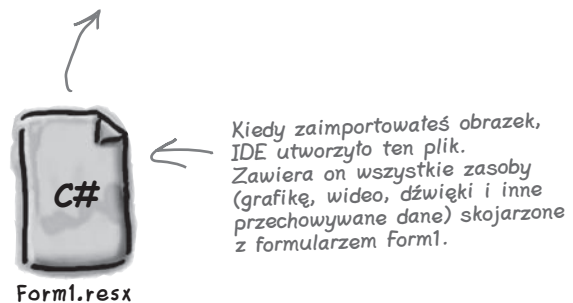
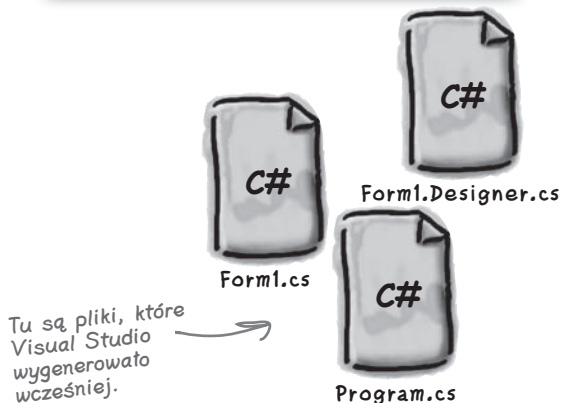
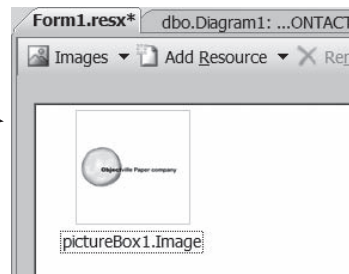
Kiedy przeciągasz kontrolkę PictureBox na formularz, IDE automatycznie tworzy plik Form1.resx, aby przechowywać zasoby i trzymać je razem w projekcie. Kliknij dwukrotnie ten plik, a zobaczysz niedawno zaimportowany obrazek.



Ten obrazek jest teraz w zasobach aplikacji listy kontaktowej.



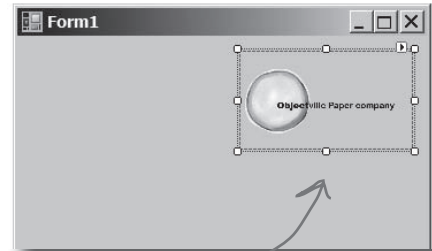
Gdy klikniesz Form1.resx, w oknie Solution Explorer możesz zobaczyć zaimportowane logo. Plik łączy obrazek z kontrolką PictureBox, natomiast IDE dodaje odpowiedni kod, niezbędny do takiego powiązania.



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.

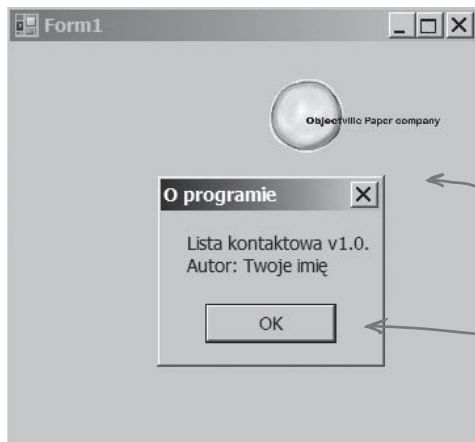
P: Co robi jakieś \n w kodzie?

O: To symbol łamania linii. W ten sposób mówimy C#, aby umieścić 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 napisalesz.

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?

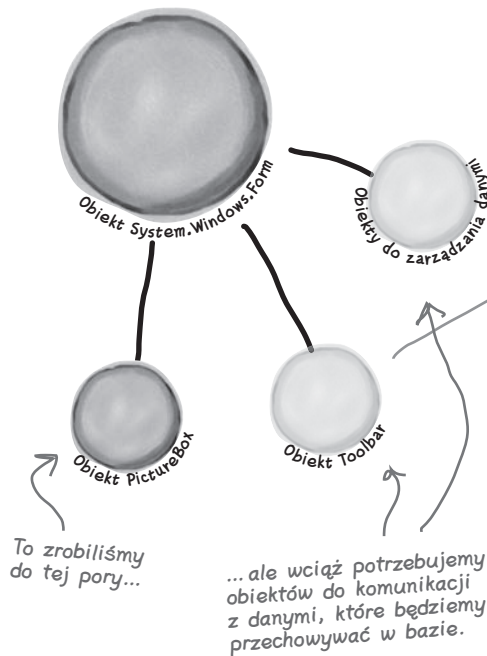
O: Przycisk „Stop Debugging” znajduje się na specjalnym pasku narzędzi, który staje się widoczny dopiero po uruchomieniu programu. Uruchom aplikację jeszcze raz i sprawdź, czy przycisk się pojawił.

Co zrobiliśmy do tej pory

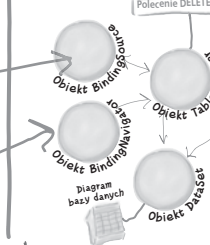
Stworzyliśmy formularz i wstawiliśmy kontrolkę PictureBox, która wyświetla komunikat w momencie kliknięcia. Kolejnym zadaniem będzie dodanie wszystkich innych pól karty, takich jak nazwa kontaktu i numer telefonu.

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.

Obiekty wizualne .NET



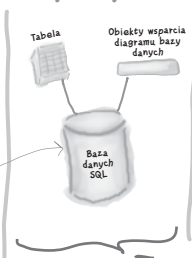
Obiekty bazy danych .NET



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



Pamięć bazy danych



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

Paczka instalatora



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

Potrzebujemy bazy danych do przechowywania naszych informacji

Zanim dodamy 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.

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

1 Dodaj nową bazę danych SQL do projektu.

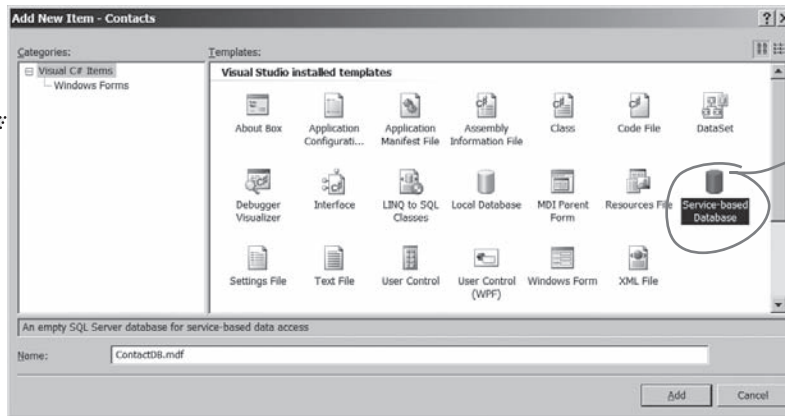
W oknie Solution Explorer kliknij prawym przyciskiem myszy projekt **Contacts**, wybierz Add, a następnie New Item. Kliknij ikonę SQL Database, a jako nazwę wpisz **ContactDB.mdf**.

→ 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**, jeżeli 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.



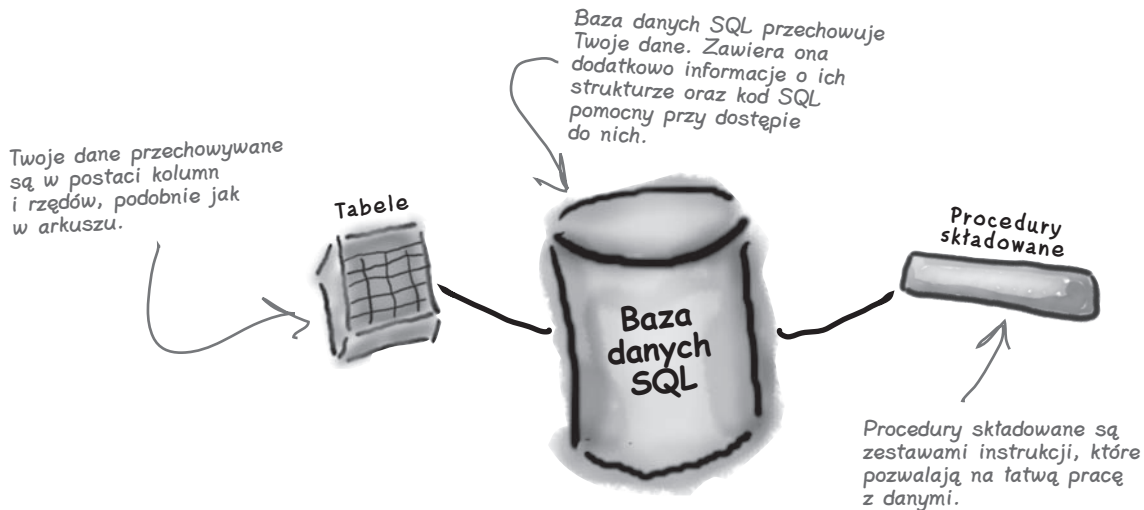
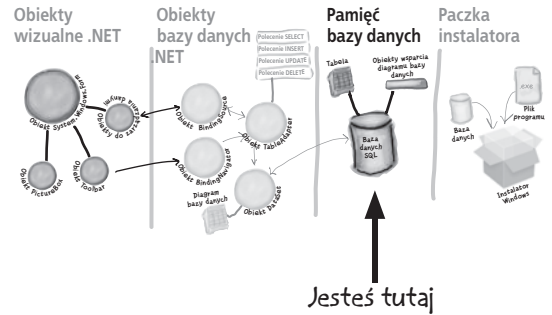
Obejrzyj to!

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

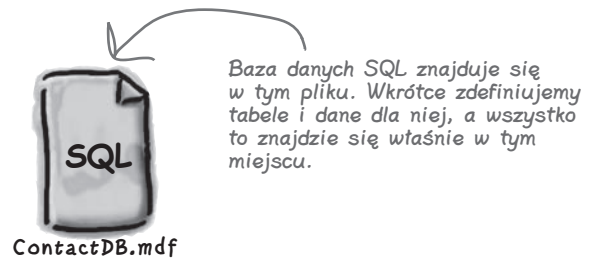
Kiedy nakazałeś IDE dodać nową bazę SQL do projektu, IDE ją stworzyło. **Baza danych SQL** to system, który przechowuje dane w sposób zorganizowany i relacyjny. Dodatkowo IDE udostępnia wszystkie narzędzia niezbędne do pracy z nimi.

Dane w bazie SQL przechowywane są w tabelach. Na bieżąco 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.



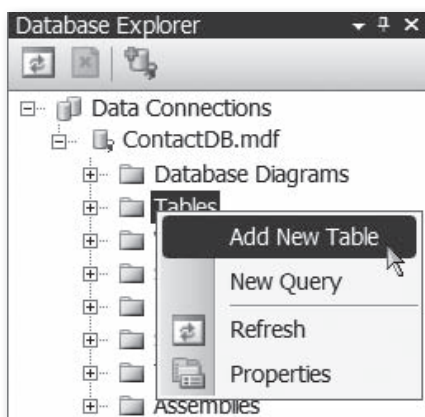
← [Notka działu marketingu: Czy możemy wstawić tutaj reklamę „Head First SQL”?]

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 utworzymy 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?

U: Kolumna to jedno pole w tabeli. A zatem w tabeli People możesz mieć kolumny FirstName oraz LastName, które będą oznaczały imię i nazwisko osoby. Pola takie zawsze będą miały typ, taki jak String, Date lub Bool.

P: Do czego jest nam potrzebna kolumna ContactID?

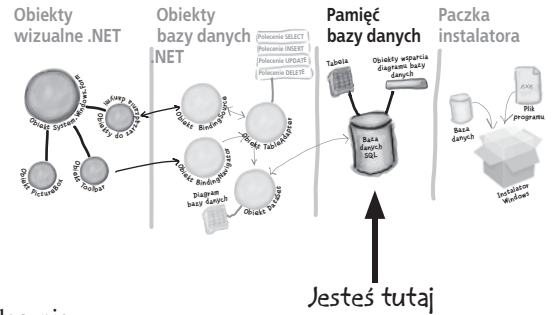
U: Umożliwia ona posiadanie unikalnego identyfikatora przez każdy rekord w większości tabel. Dlatego w przypadku przechowywania listy kontaktów do poszczególnych osób zdecydowaliśmy się stworzyć specjalną kolumnę i nazwać ją ContactID.

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?

U: 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 równoległe z tą pozycją.

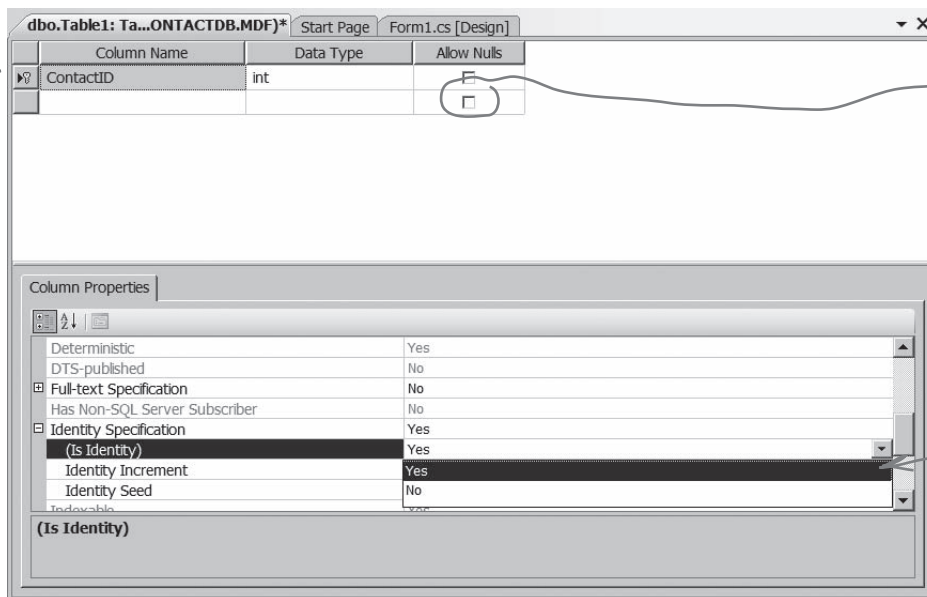


3 Powiedz bazie danych, aby automatycznie generowała identyfikatory.

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.

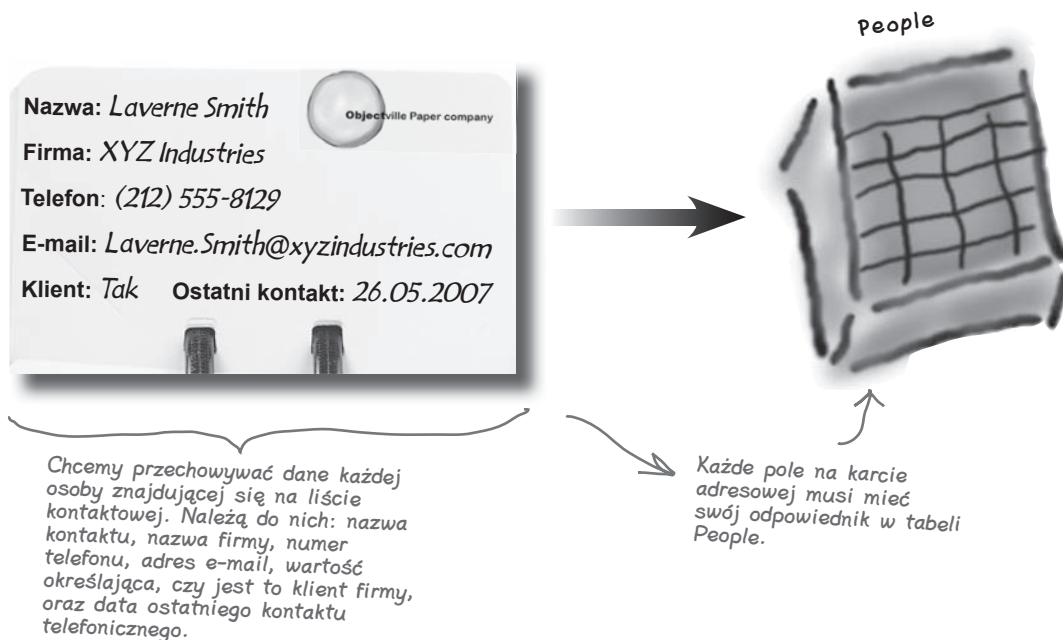


Ważne jest, abyś pozostawił to pole odznaczone. Dopóki klucz główny jest podstawowym sposobem wyszukiwania rekordów, dopóty pole ContactID musi mieć wartość.

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.



WYTEŻ UMYSŁ

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

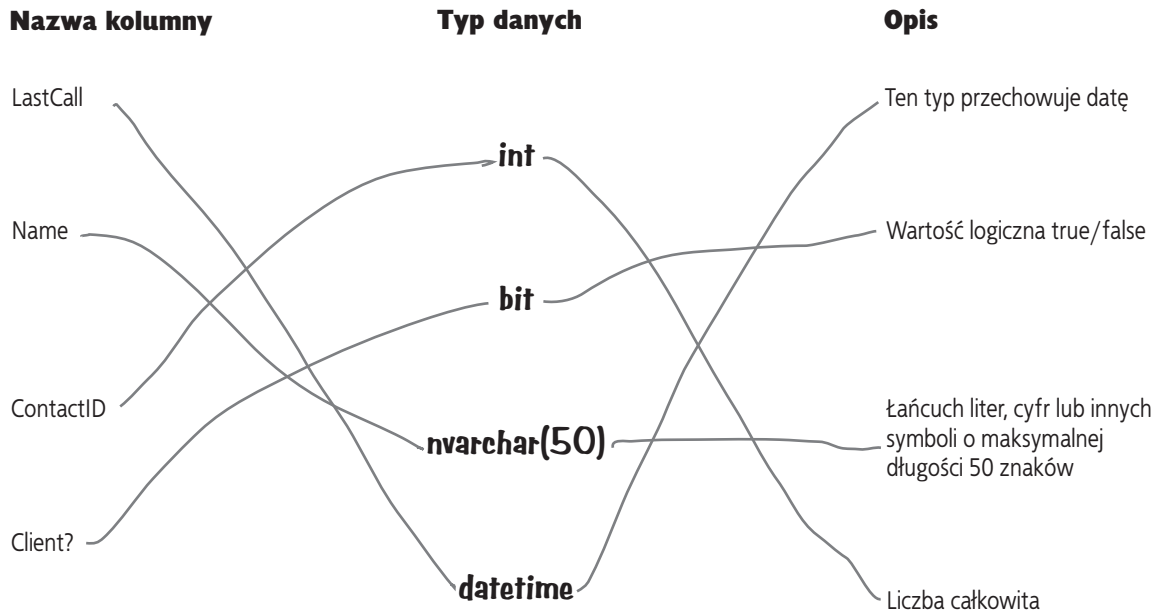
* * * KTO CO ROBI? ?

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

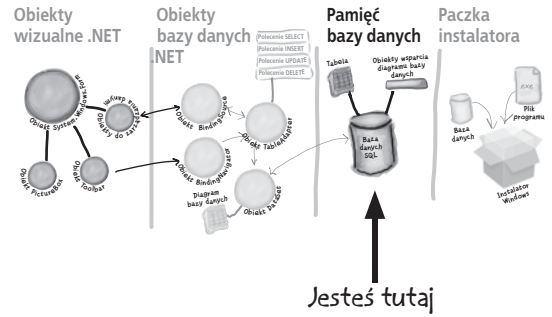
* * ? * KTO CO ROBI?

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:



Column Name	Data Type	Allow Nulls
ContactID	int	<input type="checkbox"/>
Name	nvarchar(50)	<input checked="" type="checkbox"/>
Company	nvarchar(50)	<input checked="" type="checkbox"/>
Telephone	nvarchar(50)	<input checked="" type="checkbox"/>
Email	nvarchar(50)	<input checked="" type="checkbox"/>
Client	bit	<input checked="" type="checkbox"/>
LastCall	datetime	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

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.

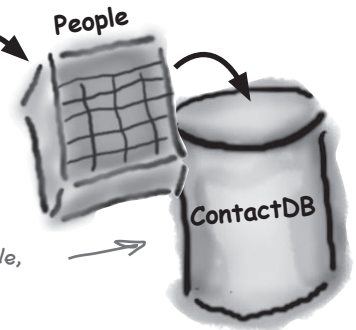
Choose Name

Enter a name for the table:

OK Cancel

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.

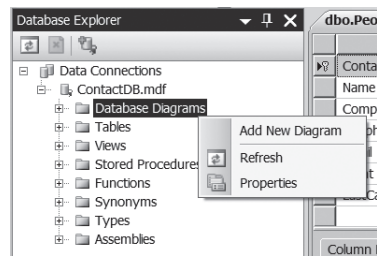


Utwórz diagram dla swoich danych, aby aplikacja miała do nich dostęp

Stworzyłeś bazę danych i tabelę. Nadszedł już czas, aby aplikacja się o tym dowiedziała. To jest właśnie moment, w którym diagram wracza 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.

1 Stwórz nowy diagram bazy danych.

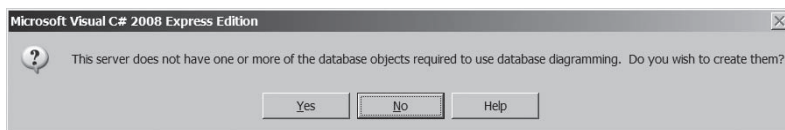
Przejdź do okna Database Explorer i kliknij prawym przyciskiem myszy element Database Diagrams. Wybierz Add New Diagram.



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

2 Pozwól IDE wygenerować kod dostępu.

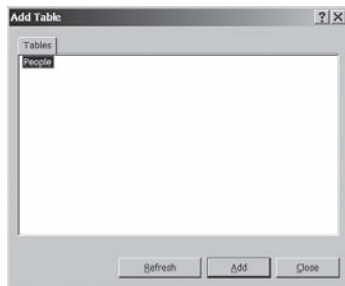
Zanim IDE dowie się o konkretnej tabeli, musi najpierw wygenerować kilka podstawowych procedur składających się do komunikacji z bazą. Po prostu naciśnij Yes i pozwól IDE wykonać całą pracę.



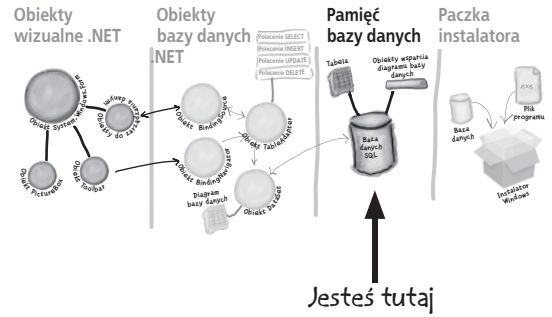
IDE tworzy kilka procedur składających, które pozwolą na interakcję kodu programu z utworzoną bazą danych.

3 Wybierz table, 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.

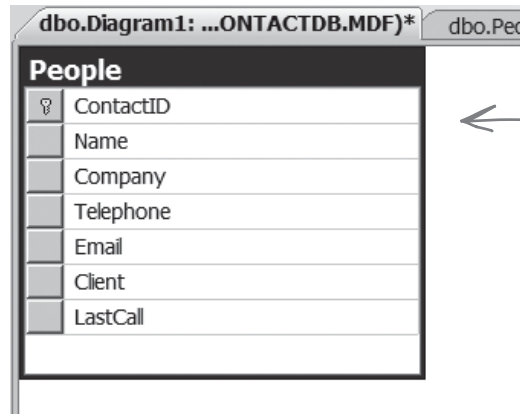


4 Nazwij diagram PeopleDiagram.

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 chciałbyś dodać do diagramu, to one także się tutaj pojawią.

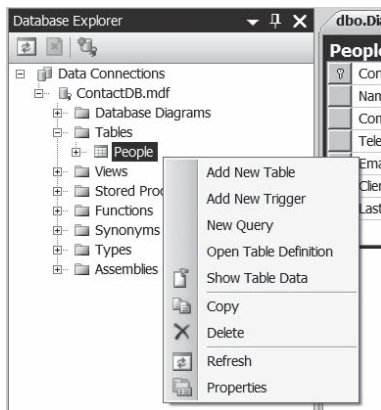
Diagram bazy danych jest dla Visual Studio IDE opisem tabeli. IDE używa go do automatycznego generowania kodu pomocnego w pracy z bazą.

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.

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.



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

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

Nazwa: Liz Nelson

Firma: JTP

Telefon: (419) 555-2578

E-mail: LizNelson@JTR.org

Klient: Tak **Ostatni kontakt:** 04.03.2006

Nazwa: Lloyd Jones

Firma: Black Box Inc.

Telefon: (718) 555-5638

E-mail: LJones@xblackboxinc.com

Klient: Tak **Ostatni kontakt:** 26.05.2007

Nazwa: Lucinda Ericson

Firma: Ericson Events

Telefon: (212) 555-9523

E-mail: Lucy@EricsonEvents.info

Klient: Nie **Ostatni kontakt:** 17.05.2007



- 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 zwykłego „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

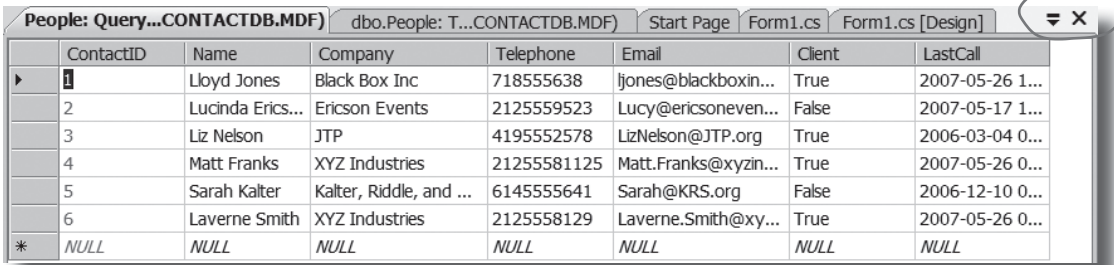
Wszystkie dane są tutaj

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.

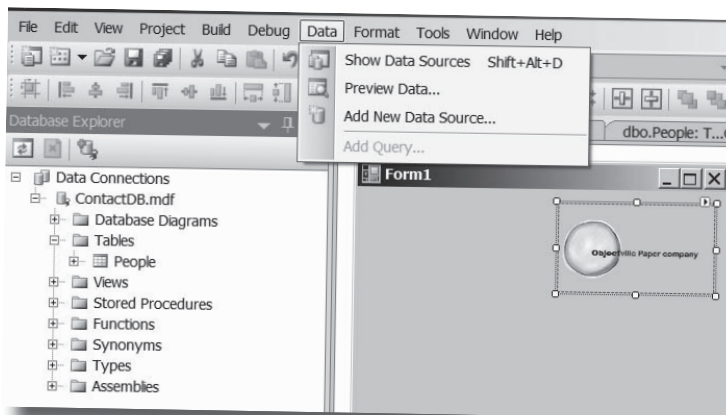
- 1 Przejdź z powrotem do formularza aplikacji.**
Zamknij tabelę People i diagram bazy danych ContactDB. Powinieneś w tej chwili widzieć zakładkę Form1.cs [Design].

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

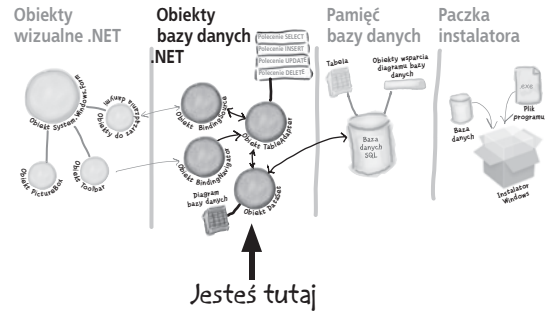


ContactID	Name	Company	Telephone	Email	Client	LastCall
1	Lloyd Jones	Black Box Inc	718555638	ljones@blackboxin...	True	2007-05-26 1...
2	Lucinda Eric...	Ericson Events	2125559523	Lucy@ericsoneven...	False	2007-05-17 1...
3	Liz Nelson	JTP	4195552578	LizNelson@JTP.org	True	2006-03-04 0...
4	Matt Franks	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

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



Źródło danych, które stworzysz, będzie miało za zadanie obsłużyć komunikację pomiędzy formularzem i bazą danych.

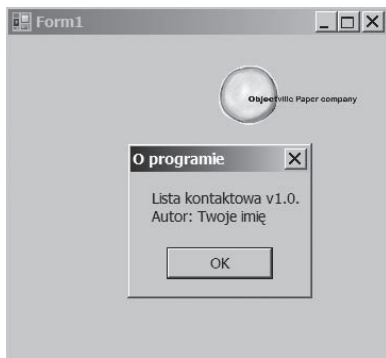


3 Skonfiguruj źródło danych.

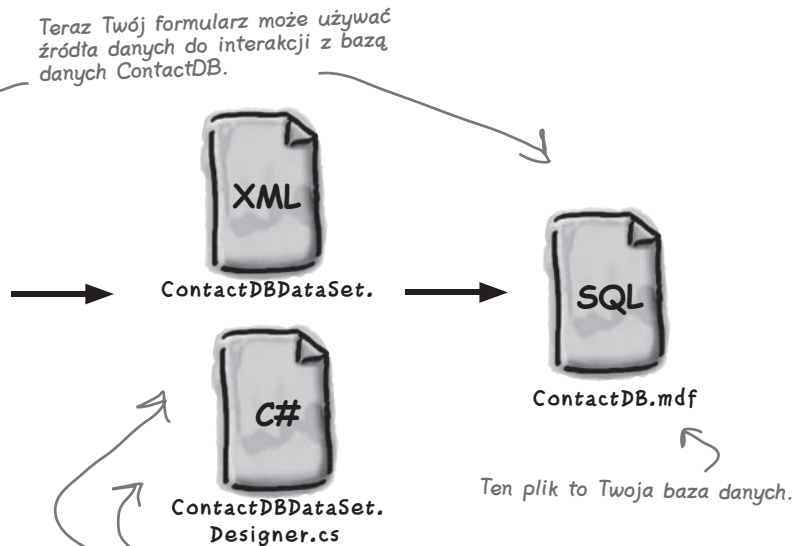
Teraz musisz dostosować źródło danych, aby używało bazy ContactDB. W tym celu powinieneś wykonać następujące czynności:

- ◆ 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.

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



To jest Twój formularz.



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

Ten plik to Twoja baza danych.

Dodaj kontrolki powiązane z bazą danych do formularza

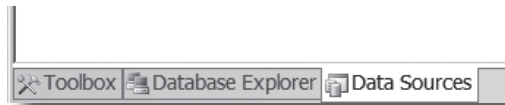
Teraz możemy powrócić do naszego formularza i dodać kilka kontrolki. Nie będą to jednak zwykłe kontrolki, tylko takie, które są *powiązane* z naszą bazą danych i kolumnami w tabeli People. Oznacza to, że każda zmiana danych w jednej z nich pociągnie za sobą automatyczną zmianę informacji w odpowiedniej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

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

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

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

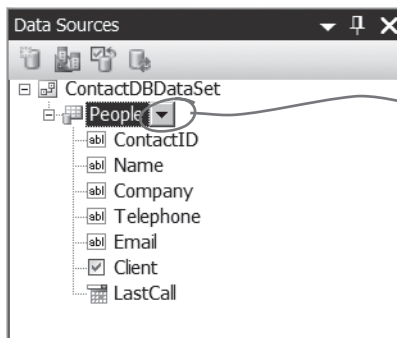
Jeżeli nie widzisz tej zakładki, wybierz „Show Data Sources” z menu Data.



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.

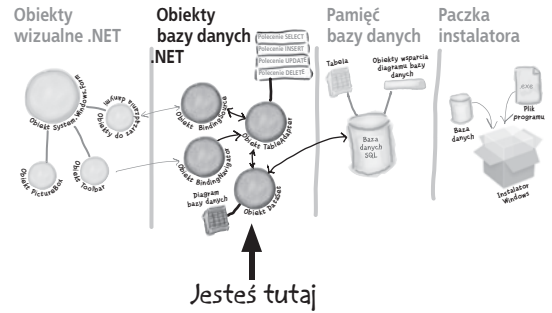
Możesz także poszukać, a następnie kliknąć zakładkę Data Sources w dolnej części okna Database Explorer.

- 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.

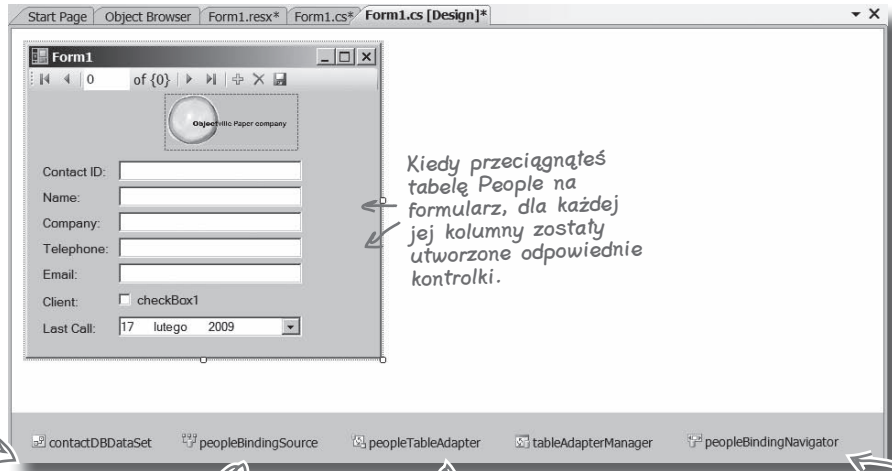


3 Utwórz kontrolki powiązane z tabelą People.

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

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

IDE tworzy ten pasek narzędzi do nawigacji po tabeli People.



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.

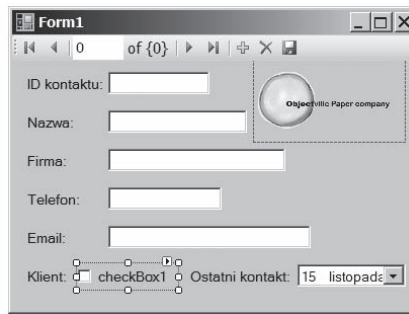
Nasz formularz będzie bardziej intuicyjny, jeżeli będzie wyglądał jak karta kontaktowa.



① Wyrównaj kontrolki i pola tekstowe.

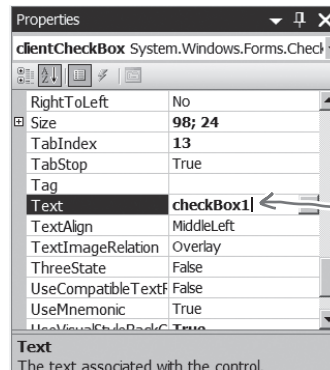
Wyrównaj kontrolki i pola tekstowe względem lewej krawędzi formularza. Będzie on wyglądał podobnie do innych aplikacji, a użytkownicy będą czuli się znacznie bardziej komfortowo podczas korzystania z programu. Przetłumacz angielskie treści etykiet, powstałych automatycznie na podstawie nazw kolumn z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

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

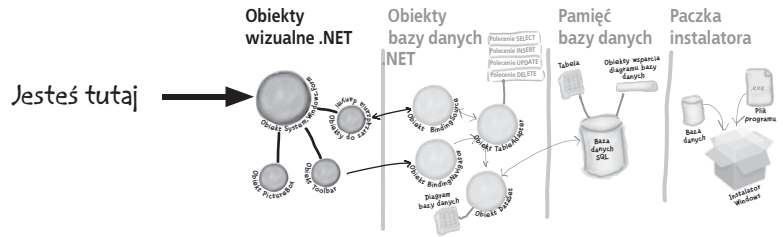


② Zmień właściwość Text kontrolki CheckBox.

Podczas wstawiania kontrolki do formularza obok kontrolki CheckBox pojawia się 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.



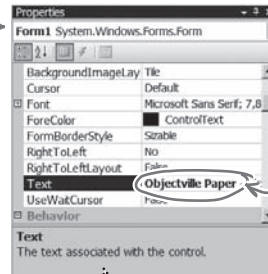
③ Nadaj aplikacji profesjonalny wygląd.

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 kontrolki 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.

Dobra, jeszcze jedna rzecz...

Jazda próbna

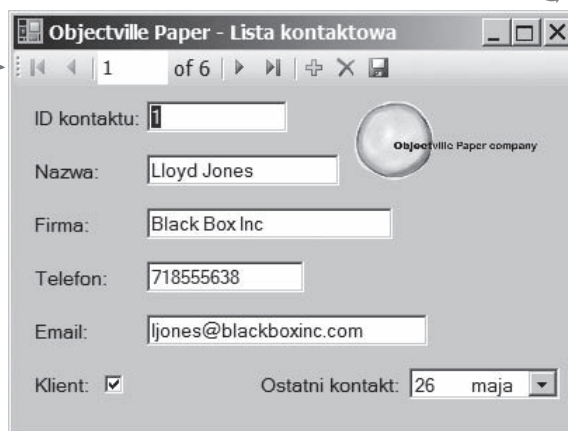
Pozostała nam do zrobienia tylko jedna rzecz... uruchomić program i sprawdzić, czy wszystko działa tak, jak powinno. Zrób to tak samo, jak do tej pory — naciśnij klawisz F5 na klawiaturze albo kliknij w przycisk z zieloną strzałką  na pasku narzędzi (lub wybierz „Run” z menu Debug).

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.

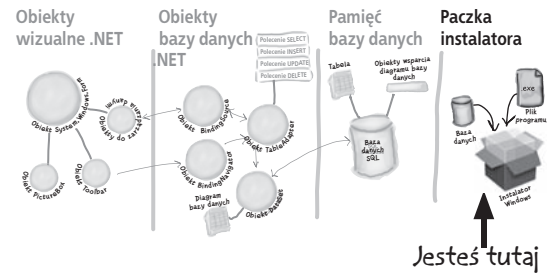
W tym przypadku plik wykonywalny oraz baza danych SQL znajdują się w katalogu bin/Debug. Plik bazodanowy jest kopiowany za każdym razem, więc zmiany wprowadzone w trakcie pracy z IDE zostaną utracone. Jeżeli plik wykonywalny będzie uruchamiany bezpośrednio z poziomu systemu Windows, dane będą zapisywane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

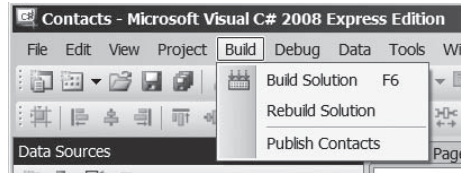
Jak zamienić TWOJĄ aplikację w aplikację WSZYSTKICH

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.

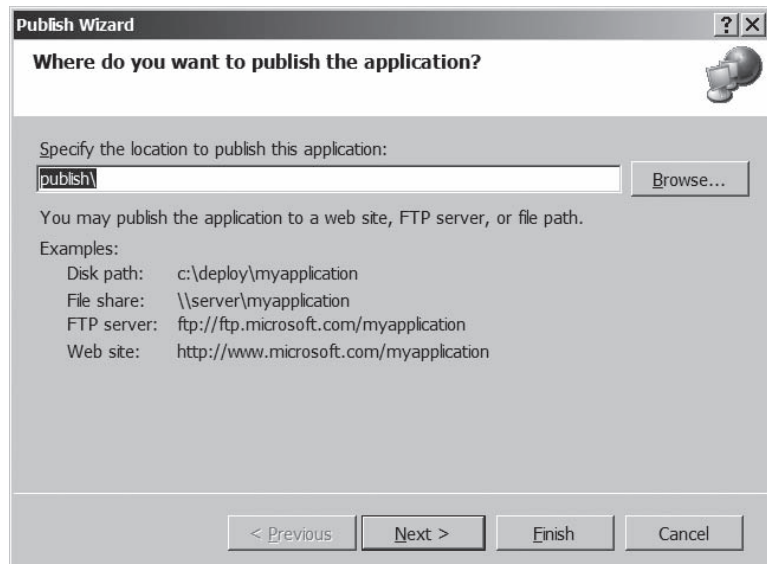


- 1 Wybierz *Publish Contacts* z menu Build.



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.

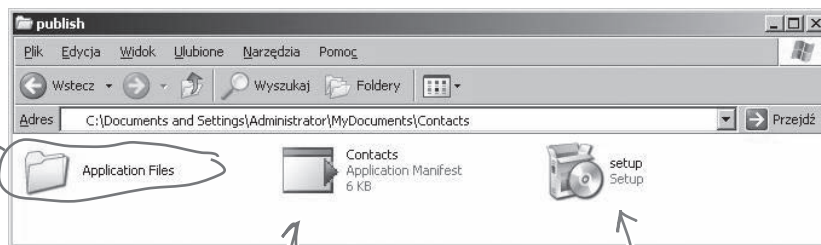
- 2 Zaakceptuj wszystkie domyślne wartości w oknie Publish Wizard poprzez naciśnięcie przycisku Finish. Zobaczysz, jak Twoja aplikacja będzie pakowana, a następnie ujrzysz folder, w którym znajdzie się plik setup.exe.



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 dotychczas 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!

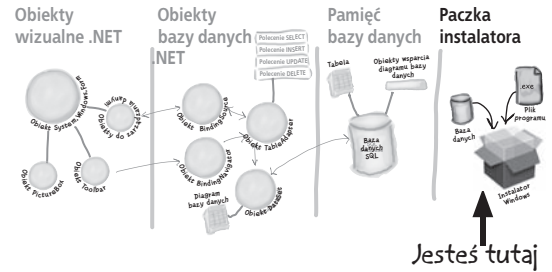


Wydaje się, że szef jest zadowolony. Dobra robota! Pozostała tylko jedna rzecz do zrobienia, zanim pofruniesz szusować na stokach, chociaż...

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.

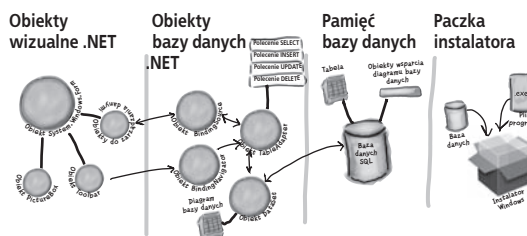
Każdy z sześciu rekordów, które na początku wpisałeś, będzie tutaj dostępny.

TESTUJ WSZYSTKO!

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

Stworzyłeś pełnowartościową aplikację bazodanową

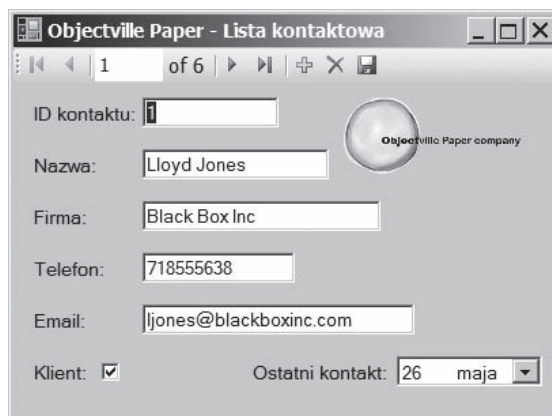
Dzięki Visual Studio IDE stworzenie aplikacji Windows, zaprojektowanie i utworzenie bazy danych oraz połączenie ich w jedną, zgrabną całość jest dość proste. Możesz nawet wygenerować program instalacyjny za pomocą zaledwie kilku kliknięć.



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.