helion.pl

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

C# 3.0 dla .NET 3.5. Księga eksperta

Autor: Joseph Mayo
Tłumaczenie: Tomasz Bienkiewicz, Jacek Janusz
ISBN: 978-83-246-2141-5
Tytuł oryginału: C# 3.0 Unleashed: With the .NET Framework 3.5
Format: 172 × 245, stron: 1112
Oprawa: twarda

Kompletne źródło wiedzy na temat C# i .NET!

- Jak tworzyć interfejs użytkownika?
- Jak uzyskać dostęp do danych z wykorzystaniem LINQ?
- Jak wdrażać napisany kod?

C# to jeden z głównych języków, który możesz wykorzystać jeżeli chcesz tworzyć rozwiązania dla platformy .NET. Jego Nnajnowsza wersja 3.0 wniosła wprowadziła wiele udoskonaleń nowości takich jak, a wśród nich: typy domniemane, typy anonimowe, uproszczone inicjowanie obiektów oraz nowe słowa kluczowe ułatwiające korzystanie z zapytań SQL. Jednak oprócz tych nowości, w książce znajdziesz również wyczerpujący opis wszystkich elementów języka C# — począwszy od składni, skończywszy na wdrażaniu kodu. Nawiązując do najlepszych tradycji tej serii „Księga eksperta" książka „C# 3.0. Księga eksperta„ stanowi kompletne źródło wiedz na temat języka C# oraz platformy .NET w wersji 3.5.

Joe Mayo podzielił książkę na dziesięć głównych części. Wśród nich znajdziesz te poświęcone podstawom języka C#. Dowiesz się zatem, co to jest tak naprawdę platforma .NET, poznasz środowisko programistyczne Visual Studio 2008 oraz zaznajomisz się z jego e elementami, składnią, wyrażeniami i instrukcjami języka C#. Część pierwsza tworzy podwaliny Twojej przygody z C# i platformą .NET. Kolejne części zawierają coraz bardziej zaawansowaną wiedzę. Szczególną uwagę warto zwrócić na te poświęcone programowaniu obiektowemu, dostępowi do danych z wykorzystaniem LINQ, tworzeniu interfejsu użytkownika czy też wdrażaniu kodu. Jeżeli jesteś programistą C# lub chcesz rozpocząć przygodę z tym językiem i platformą .NET jest to obowiązkowa pozycja w Twojej biblioteczce!

Twórz zaawansowane rozwiązania wykorzystując najlepsze narzędzia!

Kontakt

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

Spis treści

Wstęp **31**

Część I Podstawy języka C# **37**

Rozdział 1. Wprowadzenie do platformy .NET **39**
Co to jest .NET 40
Wspólne środowisko uruchomieniowe (CLR) 42
Dlaczego wspólne środowisko uruchomieniowe jest ważne? 42
Możliwości CLR 43
Proces uruchamiania CLR 43
Biblioteka klas platformy .NET (FCL) 46
C# i inne języki platformy .NET 47
Wspólny system typów (CTS) 48
Specyfikacja wspólnego języka (CLS) 49
Podsumowanie 49

Rozdział 2. Wprowadzenie do języka C# i środowiska Visual Studio 2008 ..... **51**
Budowanie prostego programu w języku C# 52
Tworzenie projektu w środowisku Visual Studio 2008 (VS2008) 56
Uruchamianie kreatora nowego projektu 57
Rozwiązania i projekty 60
Kodowanie w środowisku VS2008 60
Budowanie i uruchamianie aplikacji 63
Ustawianie opcji kompilatora 66
Komentowanie kodu 67
Komentarze wielowierszowe 67
Komentarze jednowierszowe 67
Komentarze dokumentacji w standardzie XML 68
Identyfikatory i słowa kluczowe 70
Identyfikatory 70
Słowa kluczowe 71
Konwencje i styl 73
Zmienne i typy 73
Zmienne 73
Typy proste 75
Typ łańcuchowy 79
Przypisanie oznaczone 80
Komunikacja z programami 80
Komunikacja za pomocą ekranu konsoli 81
Komunikacja za pomocą wiersza poleceń 82

Parametry wiersza poleceń w VS2008 ... 82
Zwracanie wartości z programu ... 84
Podsumowanie ... 85

Rozdział 3. Wyrażenia i instrukcje języka C# ... 87
Operatory języka C# ... 88
Operatory jednoargumentowe ... 88
Operatory dwuargumentowe ... 91
Operatory relacji ... 93
Operatory logiczne ... 95
Operatory przypisania ... 98
Operator trójargumentowy ... 98
Inne operatory ... 99
Instrukcje ... 101
Bloki i zasięg zmiennych ... 102
Etykiety ... 103
Priorytet i łączność operatorów ... 103
Instrukcje wyboru i pętli ... 104
Instrukcje if ... 104
Instrukcje switch ... 106
Pętle w języku C# ... 109
Instrukcje goto ... 113
Instrukcje break ... 114
Instrukcje continue ... 115
Instrukcje return ... 116
Podsumowanie ... 116

Rozdział 4. Typy referencyjne i wartościowe ... 117
Krótkie wprowadzenie do typów referencyjnych i wartościowych ... 118
Ujednolicony system typów ... 119
W jaki sposób działa ujednolicony system typów ... 119
Użycie typu object w programowaniu ogólnym ... 120
Wpływ opakowywania i rozpakowywania na wydajność ... 122
Przydział pamięci dla typu referencyjnego i wartościowego ... 124
Przydział pamięci dla typu referencyjnego ... 125
Przydział pamięci dla typu wartościowego ... 126
Przypisanie dla typu referencyjnego i wartościowego ... 127
Przypisanie dla typu referencyjnego ... 127
Przypisanie dla typu wartościowego ... 130
Więcej różnic między typami referencyjnymi a wartościowymi ... 131
Różnice dziedziczenia pomiędzy typami referencyjnymi a wartościowymi ... 132
Różnice konstrukcyjne i finalizacyjne pomiędzy typami referencyjnymi a wartościowymi ... 132
Rozważania dotyczące rozmiaru obiektów dla typów referencyjnych i wartościowych ... 133

Typy języka C# i środowiska .NET Framework ... 134
Zamienniki w języku C# i wspólny system typów (CTS) ... 134
Użycie typu System.Guid ... 135
Użycie typu System.DateTime ... 137
Typy dopuszczające wartości puste ... 141
Podsumowanie ... 144

Rozdział 5. Operacje na łańcuchach ... 145
Typ string języka C# ... 146
Formatowanie łańcuchów ... 147
Porównywanie łańcuchów ... 150
Sprawdzanie warunku równości łańcuchów ... 151
Łączenie łańcuchów ... 152
Kopiowanie łańcuchów ... 153
Sprawdzanie zawartości łańcucha ... 154
Wyodrębnianie informacji z łańcucha ... 154
Wyrównywanie i przycinanie wyjściowego łańcucha ... 156
Modyfikacja zawartości łańcucha ... 157
Dzielenie i łączenie łańcuchów ... 159
Operacje na znakach łańcucha ... 160
Wpływ puli wewnętrznej na obsługę łańcuchów CLR ... 161
Klasa StringBuilder ... 163
Metoda Append ... 163
Metoda AppendFormat ... 163
Metoda EnsureCapacity ... 164
Metoda ToString ... 164
Wyrażenia regularne ... 165
Podstawowe operacje na wyrażeniach regularnych ... 165
Więcej wyrażeń regularnych ... 166
Aplikacja służąca do ćwiczeń z wyrażeniami regularnymi ... 167
Podsumowanie ... 170

Rozdział 6. Użycie tablic i typów wyliczeniowych ... 171
Tablice ... 172
Tablice jednowymiarowe ... 173
Tablice wielowymiarowe ... 175
Tablice postrzępione ... 176
Klasa System.Array ... 178
Zakresy tablic ... 178
Przeszukiwanie i sortowanie ... 179
Użycie typów wyliczeniowych ... 180
Struktura System.Enum ... 184
Przekształcenia między typami wyliczeniowymi, całkowitymi i łańcuchowymi ... 184
Iteracyjne przetwarzanie elementów typu wyliczeniowego ... 185
Inne elementy struktury System.Enum ... 186
Podsumowanie ... 187

Rozdział 7. Debugowanie aplikacji za pomocą Visual Studio 2008 ... 189
Krokowe uruchamianie programu ... 190
Program demonstrujący działanie debugera ... 190
Ustawianie punktów wstrzymania ... 191
Kontrola stanu programu ... 192
Krokowe uruchamianie programu ... 194
Inne przydatne polecenia ułatwiające debugowanie ... 195
Użycie debugera w celu odnalezienia błędu w programie ... 196
Podłączanie do procesów ... 200
Podsumowanie ... 203

Część II Programowanie zorientowane obiektowo w języku C# ... 205

Rozdział 8. Projektowanie obiektów ... 207
Elementy obiektu ... 208
Elementy statyczne i instancyjne ... 209
Pola ... 210
Pola stałe ... 210
Pola readonly ... 211
Metody ... 211
Właściwości ... 212
Deklarowanie właściwości ... 212
Użycie właściwości ... 213
Właściwości automatyczne ... 213
Gotowy fragment kodu dla właściwości w środowisku VS2008 ... 214
Indeksatory ... 215
Gdzie mogą zostać użyte typy częściowe? ... 216
Klasy statyczne ... 217
Klasa System.Object ... 217
Sprawdzanie typu obiektu ... 217
Porównywanie referencji ... 218
Sprawdzanie równości ... 218
Uzyskiwanie wartości mieszających ... 219
Klonowanie obiektów ... 219
Używanie obiektów jako łańcuchów ... 220
Podsumowanie ... 221

Rozdział 9. Implementacja reguł zorientowanych obiektowo ... 223
Dziedziczenie ... 224
Klasy bazowe ... 225
Wywoływanie elementów klasy bazowej ... 226
Ukrywanie elementów klasy bazowej ... 227
Obsługa wersji ... 227
Klasy opieczętowane ... 230
Hermetyzacja organizacji wewnętrznej obiektu ... 231
Ukrywanie danych ... 231
Modyfikatory wspierające hermetyzację ... 232

Modyfikatory dostępu do obiektów ... 235
Zawieranie i dziedziczenie ... 236
Polimorfizm ... 237
Rozpoznawanie problemów rozwiązywanych przez polimorfizm ... 238
Rozwiązywanie problemów za pomocą polimorfizmu ... 241
Właściwości polimorficzne ... 243
Indeksatory polimorficzne ... 244
Przesłanianie elementów klasy System.Object ... 245
Podsumowanie ... 247

Rozdział 10. Metody kodowania i operatory tworzone przez użytkownika ... 249
Metody ... 250
Definiowanie metod ... 250
Zmienne lokalne ... 251
Parametry metod ... 253
Przeciążanie metod ... 260
Przeciążanie operatorów ... 262
Przeciążanie operatorów matematycznych dla typów tworzonych przez użytkownika ... 262
Przeciążanie operatorów logicznych dla typów tworzonych przez użytkownika ... 265
Inne wskazówki związane z przeciążaniem operatorów ... 266
Konwersje i przeciążanie operatorów konwersji ... 268
Konwersje niejawne i jawne ... 268
Operatory konwersji typów wartościowych stworzonych przez użytkownika ... 271
Operatory konwersji typów referencyjnych stworzonych przez użytkownika ... 275
Metody częściowe ... 276
Metody rozszerzające ... 278
Podsumowanie ... 279

Rozdział 11. Obsługa błędów i wyjątków ... 281
Dlaczego używa się obsługi wyjątków? ... 282
Składnia procedury obsługi wyjątku: podstawowy blok try/catch ... 283
Zapewnianie zwalniania zasobów przy użyciu bloków finally ... 285
Obsługa wyjątków ... 286
Obsługa różnych typów wyjątków ... 286
Obsługa i przekazywanie wyjątków ... 287
Powrót ze stanu wyjątku ... 290
Tworzenie wyjątków przez użytkownika ... 293
Instrukcje checked i unchecked ... 295
Podsumowanie ... 297

Rozdział 12. Programowanie oparte na zdarzeniach: obiekty delegowane i zdarzenia ... 299
Udostępnianie delegacji ... 301
Definiowanie delegacji ... 301
Tworzenie metod obsługujących delegacje ... 302
Łączenie delegacji i metod obsługujących ... 302

Wykonywanie metod poprzez delegacje ... 303
Delegacje wielozakresowe ... 303
Sprawdzanie równości delegacji ... 306
Implementacja wnioskowania delegacji ... 307
Przypisywanie metod anonimowych ... 307
Kodowanie zdarzeń ... 309
Definiowanie procedur obsługi zdarzeń ... 310
Rejestrowanie zdarzeń ... 311
Implementacja zdarzeń ... 312
Uruchamianie zdarzeń ... 314
Modyfikacja metod zdarzeń Add i Remove ... 316
Podsumowanie ... 321

Rozdział 13. Nazewnictwo i organizacja typów w przestrzeniach nazw ... 323
Dlaczego przestrzenie nazw muszą istnieć? ... 324
Organizowanie kodu ... 325
Unikanie konfliktów ... 325
Dyrektywy przestrzeni nazw ... 326
Dyrektywa using ... 326
Dyrektywa alias ... 327
Tworzenie przestrzeni nazw ... 329
Składowe przestrzeni nazw ... 333
Zasięg i widoczność ... 333
Kwalifikatory związane z synonimem przestrzeni nazw ... 335
Synonimy zewnętrznych przestrzeni nazw ... 336
Podsumowanie ... 338

Rozdział 14. Implementacja klas abstrakcyjnych i interfejsów ... 339
Klasy abstrakcyjne ... 340
Różnice między klasami abstrakcyjnymi a interfejsami ... 343
Implementacja interfejsów ... 343
Definiowanie typów interfejsów ... 344
Metody ... 345
Właściwości ... 345
Indeksatory ... 345
Zdarzenia ... 346
Implementacja niejawna ... 346
Implementacja interfejsu dla pojedynczej klasy ... 346
Symulowanie zachowania polimorficznego ... 350
Implementacja jawna ... 355
Odwzorowanie interfejsu ... 361
Dziedziczenie interfejsu ... 363
Podsumowanie ... 365

Część III Używanie zaawansowanych funkcji języka C# 367

Rozdział 15. Zarządzanie czasem życia obiektu 369
Inicjalizacja obiektów 370
Konstruktory instancyjne 371
Przeciążanie konstruktorów 372
Konstruktory domyślne 374
Konstruktory prywatne 374
Dziedziczenie i kolejność konkretyzacji 375
Konstruktory statyczne 379
Inicjalizatory obiektów 380
Finalizacja obiektów 381
Automatyczne zarządzanie pamięcią 382
Przydzielanie pamięci 383
Wewnętrzna organizacja mechanizmu oczyszczania pamięci 384
Optymalizacja mechanizmu oczyszczania pamięci 385
Właściwe zwalnianie zasobów 386
Problemy z finalizatorami 387
Wzorzec Dispose 387
Instrukcja using 389
Współpraca z mechanizmem oczyszczania pamięci 390
Sterowanie obiektami 390
Podsumowanie 392

Rozdział 16. Deklarowanie atrybutów i testowanie kodu za pomocą mechanizmów refleksji 393
Użycie atrybutów 394
Użycie pojedynczego atrybutu 395
Użycie wielu atrybutów 396
Użycie parametrów atrybutów 396
Parametry pozycyjne 397
Parametry nazwane 398
Obiekty docelowe atrybutu 398
Tworzenie własnych atrybutów 400
Atrybut AttributeUsage 400
Użycie mechanizmu refleksji 404
Uzyskiwanie informacji o programie 404
Wykorzystanie refleksji dla atrybutów 410
Dynamiczne aktywowanie kodu 411
Tworzenie pakietów kodu w trakcie działania programu przy użyciu API Reflection.Emit 413
Podsumowanie 417

Rozdział 17. Parametryzowanie typów poprzez szablony klas i tworzenie iteratorów ... 419
Kolekcje bezszablonowe ... 420
Korzyści ze stosowania szablonów ... 421
Problemy rozwiązywane przez stosowanie szablonów ... 422
Szablony są zorientowane obiektowo ... 425
Dokonywanie wyboru między tablicami, kolekcjami bezszablonowymi i kolekcjami szablonowymi ... 426
Tworzenie typów szablonowych ... 428
Implementacja listy jednokierunkowej za pomocą szablonów ... 428
Używanie szablonów poza kolekcjami ... 436
Definiowanie typu za pomocą szablonów ... 439
Implementacja iteratorów ... 443
Iterator GetEnumerator ... 444
Iteratory metod ... 446
Iteratory właściwości ... 446
Iteratory indeksatorów ... 447
Iterator operatora ... 449
Iteratory jako ciągi wartości ... 450
Zwalnianie iteratorów ... 451
Podsumowanie ... 452

Rozdział 18. Wyrażenia lambda i drzewa wyrażeń ... 453
Wyrażenia lambda ... 454
Składnia wyrażeń lambda ... 454
Użycie wyrażeń lambda ... 455
Delegacje i wyrażenia lambda ... 456
Drzewa wyrażeń ... 461
Przekształcanie wyrażenia lambda na drzewo wyrażeń ... 461
Przekształcanie drzewa wyrażeń na wyrażenie lambda ... 462
Podsumowanie ... 463

Część IV Dostęp do danych przy użyciu LINQ i platformy .NET ... 465

Rozdział 19. Dostęp do danych z wykorzystaniem LINQ ... 467
Technologia LINQ to Objects ... 469
Podstawowa składnia LINQ ... 469
Wyodrębnianie projekcji ... 470
Filtrowanie danych ... 471
Sortowanie wyników zapytania ... 472
Grupowanie danych ... 472
Łączenie danych ... 472
Tworzenie hierarchii za pomocą grupowania połączeń ... 473
Wykonywanie zapytań do baz relacyjnych za pomocą technologii LINQ to SQL ... 474
Definiowanie kontekstu danych DataContext ... 474
Zapytania przy użyciu DataContext ... 478

Modyfikacja obiektów DataContext ... 478
Wywołanie procedur składowanych ... 480
Użycie funkcji SQL ... 481
Modyfikowanie bazy danych za pomocą procedur składowanych ... 481
Modyfikacja logiki obsługi danych poprzez użycie metod częściowych ... 484
Standardowe operatory zapytań ... 488
Operatory sortujące ... 488
Operatory ustawiania ... 489
Operatory filtrujące ... 491
Operatory kwantyfikatorów ... 492
Operatory projekcji ... 492
Operatory partycjonowania ... 493
Operatory łączenia ... 494
Operatory grupowania ... 495
Operatory generujące ... 495
Operatory równości ... 496
Operatory elementarne ... 497
Operatory konwersji ... 498
Operator wiązania ... 498
Operatory agregacji ... 499
Podsumowanie ... 500

Rozdział 20. Zarządzanie danymi z wykorzystaniem ADO.NET ... 501
Architektura ADO.NET ... 502
Komponenty ADO.NET ... 502
Tryby otwartego i zamkniętego połączenia ... 504
Dostawcy danych ... 505
Wykonywanie połączeń ... 507
Przeglądanie danych ... 508
Modyfikacja danych ... 512
Wstawianie danych ... 512
Aktualizacja danych ... 512
Usuwanie danych ... 513
Wywoływanie procedur składowanych ... 514
Obsługa danych w trybie autonomicznym ... 514
Wczytywanie danych do obiektu DataSet ... 515
Zapisywanie modyfikacji DataSet do bazy danych ... 516
Użycie LINQ to DataSet ... 519
Obiekty DataTable jako źródła danych ... 520
Dostęp do pól przy zachowaniu ścisłej kontroli typów ... 520
Podsumowanie ... 521

Rozdział 21. Przetwarzanie danych w formacie XML ... 523
Przesyłanie strumieniowe danych XML ... 524
Zapisywanie danych XML ... 525
Odczytywanie danych XML ... 527

Użycie XML DOM ... 528
Odczytywanie dokumentu XML przy użyciu XPathDocument ... 529
Modyfikacja dokumentu XML przy użyciu XmlDocument ... 530
Prostszy sposób przetwarzania danych przy wykorzystaniu LINQ to XML ... 531
Obiekty LINQ to XML ... 531
Tworzenie dokumentów XML ... 531
Obsługa przestrzeni nazw dla LINQ to XML ... 533
Odczytywanie dokumentów XML ... 534
Wykonywanie zapytań dla dokumentów XML ... 534
Modyfikacja dokumentów XML ... 535
Podsumowanie ... 536

Rozdział 22. Dostęp do danych za pomocą ADO.NET Entity Framework ... 537
Encje ... 539
Tworzenie modelu EDM (Entity Data Model) w Visual Studio 2008 ... 539
Tworzenie zapytań za pomocą Entity SQL ... 543
Dostęp do encji ... 543
Wybieranie danych z encji ... 544
Tworzenie własnych encji ... 545
Schematy i odwzorowania ... 546
Dodawanie własnych encji ... 547
Wykorzystanie implementacji LINQ to Entities ... 550
Kwerendy do encji ... 550
Modyfikowanie danych encji ... 551
Podsumowanie ... 552

Rozdział 23. Dostęp do danych w sieci za pośrednictwem usług ADO.NET Data Services ... 555
Dodanie usług ADO.NET Data Services do projektu ... 556
Dostęp do usług ADO.NET Data Services za pośrednictwem HTTP i URI ... 558
Wyświetlanie zestawów encji ... 558
Wybieranie elementów encji ... 558
Filtrowanie wyników ... 561
Sortowanie encji ... 563
Używanie powiązań encji ... 563
Tworzenie kodu z biblioteką ADO.NET Data Services Client Library ... 565
Tworzenie projektu klienta ... 565
Wykonywanie zapytań do encji za pomocą WebDataQuery ... 565
Dodawanie encji ... 567
Aktualizacja encji ... 568
Usuwanie encji ... 569
Tworzenie zapytań z wykorzystaniem LINQ dla usług danych ... 569
Wykorzystanie klas wygenerowanych za pomocą narzędzia DataSvcUtil.exe ... 570
Podsumowanie ... 571

Część V Tworzenie interfejsów użytkownika 573

Rozdział 24. Interfejs użytkownika w aplikacjach konsolowych 575
Aplikacja PasswordGenerator 576
Komunikacja programu z użytkownikiem 577
Obsługa z wiersza poleceń 578
Dodawanie koloru i pozycjonowanie elementów w oknie konsoli 579
Podsumowanie 582

Rozdział 25. Tworzenie aplikacji w oparciu o formularze Windows Forms 583
Formularze Windows Forms — informacje podstawowe 584
Tworzenie aplikacji Windows Forms Application w VS2008 588
Wizualne projektowanie interfejsu w środowisku VS2008 588
Pliki aplikacji Windows Forms Application 590
Środowisko Windows Forms Designer 590
Kontrolki dostępne w Windows Forms 597
Kontrolki MenuStrip, StatusStrip i ToolStrip 600
Prezentacja danych za pomocą kontrolek DataGrid i DataBind 601
Przygotowanie projektu dla prezentacji danych 602
Wyświetlanie danych za pomocą kontrolki ListBox 603
Wyświetlanie danych za pomocą kontrolki DataGridView 603
Podstawy GDI+ 605
Obiekty Brush, Pen, Graphics — pędzel, ołówek i rysunek 605
Wyświetlanie tekstu i czcionki 606
Pozostałe okna dialogowe 608
Okna modalne i niemodalne 608
Komunikacja między oknami 610
Pozostałe predefiniowane okna dialogowe 612
Podsumowanie 614

Rozdział 26. Tworzenie aplikacji Windows Presentation Foundation (WPF) 615
Język XAML 616
Wprowadzenie do aplikacji WPF 617
Podstawy XAML 618
Kontrolki w XAML 619
Rozmieszczanie elementów w tworzonym oknie 621
Rozmieszczanie kontrolek i określanie ich rozmiarów 621
Powierzchnia Canvas 622
Powierzchnia WrapPanel 623
Powierzchnia StackPanel 623
Powierzchnia UniformGrid 624
Powierzchnia Grid 625
Powierzchnia DockPanel 628
Kontrolki WPF 629
Kontrolka Border 629
Kontrolka Button 630

Kontrolka CheckBox ... 630
Kontrolka ComboBox ... 630
Kontrolka ContentControl ... 630
Kontrolka DockPanel ... 631
Kontrolka DocumentViewer ... 631
Kontrolka Ellipse ... 632
Kontrolka Expander ... 632
Kontrolka Frame ... 633
Kontrolka Grid ... 633
Kontrolka GridSplitter ... 633
Kontrolka GroupBox ... 634
Kontrolka Image ... 634
Kontrolka Label ... 634
Kontrolka ListBox ... 635
Kontrolka ListView ... 635
Kontrolka MediaElement ... 635
Kontrolka Menu ... 635
Kontrolka PasswordBox ... 636
Kontrolka ProgressBar ... 636
Kontrolka RadioButton ... 636
Kontrolka Rectangle ... 637
Kontrolka RichTextBox ... 637
Kontrolka ScrollBar ... 637
Kontrolka ScrollViewer ... 637
Kontrolka Separator ... 638
Kontrolka Slider ... 638
Kontrolka StackPanel ... 639
Kontrolka StatusBar ... 639
Kontrolka TabControl ... 639
Kontrolka TextBlock ... 639
Kontrolka TextBox ... 640
Kontrolka ToolBar ... 640
Kontrolka ToolBarPanel ... 640
Kontrolka ToolBarTray ... 641
Kontrolka TreeView ... 641
Kontrolka UniformGrid ... 641
Kontrolka Viewbox ... 642
Kontrolka WindowsFormsHost ... 642
Kontrolka WrapPanel ... 643
Obsługa zdarzeń ... 643
Powiązanie kontrolek z danymi ... 644
Przekazywanie danych ... 644
Wyświetlanie listy danych ... 645
Style i formatowanie kontrolek ... 649
Podsumowanie ... 651

Część VI Projektowanie interfejsów użytkownika w oparciu o strony internetowe 653

Rozdział 27. Tworzenie aplikacji sieciowych za pomocą ASP.NET 655
Model aplikacji sieciowej 656
Wysokopoziomowy model aplikacji sieciowej 656
Gdzie znajduje się kod C# aplikacji sieciowej? 657
Skalowalność i zarządzanie stanem 657
Czas reakcji aplikacji 658
Korzyści z zastosowania ASP.NET 659
Tworzenie projektu ASP.NET w VS2008 660
Strona ASP.NET 661
Elementy formularza 661
Kod ukryty i cykl życia strony 664
Kontrolki 667
Kontrolki serwerowe 667
Kontrolki HTML 669
Zarządzanie stanem 669
Application — globalny stan aplikacji 670
Cache — przechowywanie informacji, które można aktualizować 671
Context — przechowywanie stanu pojedynczego żądania 672
Pliki cookie 672
Session — informacje użytkownika 673
ViewState — informacje o stanie strony 673
Strony wzorcowe i kontrolki użytkownika 674
Nawigacja 678
Rozmieszczenie elementów za pomocą pliku web.sitemap 679
Nawigacja za pomocą kontrolki Menu 680
Implementacja kontrolki TreeView 681
Wykorzystanie ścieżki nawigacji 684
Wykorzystywanie tematów 685
Tworzenie tematu 685
Tworzenie skórek 686
Tworzenie arkuszy stylów 687
Zabezpieczanie witryny 688
Prezentacja danych 691
Tworzenie obiektu biznesowego 691
Przykład powiązania danych 692
Powiązanie danych za pomocą kontrolki ObjectDataSource 693
Podsumowanie 695

Rozdział 28. Wykorzystanie ASP.NET AJAX w aplikacjach sieciowych 697
Czym jest AJAX? 698
Tworzenie witryny z wykorzystaniem ASP.NET AJAX 699
Cykl życia strony AJAX 700

Wykorzystywanie bibliotek skryptowych ... 701
Kontrolki ASP.NET AJAX ... 703
Kontrolka UpdatePanel ... 704
Kontrolka UpdateProgress ... 705
Kontrolka Timer ... 706
Dostęp do kontrolek z poziomu kodu JavaScript ... 707
Kontrolki z identyfikatorami prostymi ... 707
Kontrolki z identyfikatorami złożonymi ... 709
Wywołanie usługi sieciowej z użyciem ASP.NET AJAX ... 714
Za i przeciw wykorzystaniu technologii AJAX z usługami sieciowymi ... 714
Wykorzystanie technologii AJAX z usługami sieciowymi ... 715
Podsumowanie ... 718

Rozdział 29. Tworzenie zaawansowanych aplikacji sieciowych za pomocą Silverlight ... 721
Z czego składa się Silverlight? ... 722
Miejsce WPF i XAML w technologii Silverlight ... 722
Zależności pomiędzy Silverlight a ASP.NET, JavaScript i AJAX ... 723
Projekty Silverlight w środowisku VS2008 ... 723
Tworzenie projektu Silverlight ... 724
Elementy projektu Silverlight ... 724
Obsługa zdarzeń w aplikacji Silverlight ... 728
Kod obsługi zdarzenia kontrolki Silverlight ... 730
Silverlight i źródła danych ... 731
Prezentacja plików multimedialnych ... 734
Wykorzystanie kontrolki MediaPlayer w formularzu aplikacji sieciowej ... 734
Zarządzanie kontrolką MediaElement z poziomu języka C# ... 736
Animacja elementów interfejsu użytkownika ... 738
Podsumowanie ... 740

Część VII Komunikacja za pomocą technologii dostępnych w .NET ... 741

Rozdział 30. Technologie komunikacji sieciowej w .NET ... 743
Implementacja programu z wykorzystaniem gniazd ... 744
Program serwera ... 744
Program klienta ... 747
Implementacja programu z wykorzystaniem protokołu HTTP ... 751
Transfer plików za pomocą protokołu FTP ... 753
Umieszczanie plików w serwerze FTP ... 753
Pobieranie plików z serwera FTP ... 755
Wysyłanie wiadomości za pomocą protokołu SMTP ... 757
Sposób na szybkie wysłanie wiadomości e-mail ... 757
Wysyłanie wiadomości z załącznikami ... 758
Podsumowanie ... 758

Rozdział 31. Tworzenie usług dla systemu Windows ... 761
Tworzenie projektu usługi w VS2008 ... 762
Kreator Windows Service Wizard ... 762
Elementy projektu usługi dla Windows ... 762
Tworzenie kodu usług dla Windows ... 765
Przesłaniane metody w usługach dla Windows ... 765
Implementacja metod usługi ... 767
Konfiguracja usługi ... 770
Instalowanie usługi w systemie Windows ... 771
Konfiguracja komponentu ServiceProcessInstaller ... 771
Konfiguracja komponentu ServiceInstaller ... 772
Wdrażanie usługi ... 773
Kontroler komunikacji z usługą ... 774
Podsumowanie ... 776

Rozdział 32. Technologia .NET Remoting ... 777
Podstawy technologii Remoting ... 778
Serwer w technologii Remoting ... 779
Klient w technologii Remoting ... 781
Uruchomienie aplikacji ... 784
Kanały ... 788
Zarządzanie życiem obiektów ... 791
Podsumowanie ... 793

Rozdział 33. Tworzenie tradycyjnych usług sieciowych ASMX ... 795
Podstawy usług sieciowych ... 796
Technologie usług sieciowych ... 796
Prosta usługa sieciowa ... 797
Wyświetlanie informacji o usłudze sieciowej ... 798
Wykorzystywanie usług sieciowych ... 802
Podsumowanie ... 807

Rozdział 34. Tworzenie usług sieciowych z wykorzystaniem WCF ... 809
Tworzenie aplikacji WCF w VS2008 ... 810
Kontrakt usługi sieciowej ... 812
Interfejs usługi WCF ... 812
Deklaracja atrybutu ServiceContract ... 814
Deklaracja atrybutów OperationsContract ... 815
Tworzenie kontraktów danych ... 815
Implementacja logiki usługi sieciowej ... 817
Konfigurowanie usługi sieciowej ... 819
Element service ... 820
Element endpoint (punkt końcowy) ... 820
Element behavior ... 821
Wykorzystywanie usługi sieciowej ... 822
Referencja do usługi ... 822
Tworzenie kodu aplikacji klienta w celu wywołania usługi sieciowej ... 823
Podsumowanie ... 824

Część VIII Architektura i projektowanie aplikacji 825

Rozdział 35. Kreator klas — Visual Studio 2008 Class Designer 827
Wizualizacja kodu 828
Prezentacja obiektów 828
Prezentacja związków, dziedziczenia i interfejsów 831
Tworzenie modelu obiektowego za pomocą narzędzia Class Designer 834
Podsumowanie 839

Rozdział 36. Wzorce projektowe w C# 841
Przegląd wzorców projektowych 842
Wzorzec Iterator 842
Implementacja interfejsu IEnumerable 843
Implementacja interfejsu IEnumerator 844
Wykorzystanie iteratora 849
Niezwykłe zachowanie pętli foreach 850
Uproszczenie wzorca Iterator z wykorzystaniem iteratorów C# 853
Wzorzec Proxy 854
Przykład wzorca Proxy 855
Użycie obiektu Proxy 857
Wzorzec Template 858
Wykorzystanie wzorca Template w .NET Framework 859
Przykład implementacji wzorca Template 860
Podsumowanie 863

Rozdział 37. Tworzenie systemów wielowarstwowych 865
Problemy związane z technologią RAD 866
Aplikacja RAD w pięć minut 866
Narzędzia RAD a tworzenie wydajnych rozwiązań 867
Architektura wielowarstwowa 869
Architektura aplikacji 869
Architektura wielowarstwowa — podział logiczny 869
Architektura warstwowa — podział fizyczny 871
Podejście do architektury aplikacji 872
Przykłady architektury wielowarstwowej 872
Aplikacje wielowarstwowe, umieszczone w pojedynczych komponentach 873
Aplikacje wielowarstwowe, umieszczone w kilku komponentach 880
Podsumowanie 884

Rozdział 38. Windows Workflow 885
Tworzenie projektu aplikacji przebiegu 886
Tworzenie sekwencji przebiegu 887
Tworzenie przebiegu 887
Kod wygenerowany dla przebiegu 890
Tworzenie przebiegu stanów 891
Model przebiegu stanów wizyty lekarskiej 892
Tworzenie przebiegu stanów 892

Przekazywanie informacji pomiędzy hostem a przebiegiem za pomocą ExternalDataExchangeService ... 894
Obsługa zdarzeń w przebiegu stanów ... 899
Podsumowanie ... 903

Część IX Przegląd biblioteki .NET Framework Class Library ... 905

Rozdział 39. Zarządzanie procesami i wątkami ... 907
Zarządzanie procesami z wykorzystaniem biblioteki .NET ... 908
Uruchamianie nowego procesu ... 909
Praca z uruchomionymi procesami ... 912
Wielowątkowość ... 914
Tworzenie nowych wątków ... 914
Uruchomienie wątku — wariant uproszczony ... 915
Przekazywanie parametrów do wątków ... 915
Obiekt ThreadPool ... 916
Synchronizacja wątków ... 917
Instrukcja lock ... 917
Klasa Monitor — rzeczywista implementacja instrukcji lock ... 918
Zachowanie równowagi pomiędzy wątkami zapisu i odczytu ... 919
Podsumowanie ... 921

Rozdział 40. Tworzenie różnych wersji językowych aplikacji ... 923
Pliki zasobów ... 924
Tworzenie pliku zasobów ... 924
Zapis do pliku zasobów ... 927
Odczyt z pliku zasobów ... 928
Konwersja pliku zasobów ... 929
Tworzenie zasobów graficznych ... 931
Ustawienia regionalne ... 936
Implementacja ustawień dla wielu regionów ... 937
Wyszukiwanie zasobów ... 942
Podsumowanie ... 943

Rozdział 41. Używanie mechanizmu Interop (usługi P/Invoke i COM) oraz tworzenie kodu nienadzorowanego ... 945
Kod nienadzorowany ... 946
Znaczenie pojęcia „kod nienadzorowany” ... 947
Magia wskaźników ... 947
Operator sizeof() ... 951
Operator stackalloc ... 952
Instrukcja fixed ... 954
Usługa Platform Invoke ... 957
Komunikacja z komponentami COM w .NET ... 959
Wczesne wiązanie ... 959
Późne wiązanie ... 961
Udostępnianie komponentów środowiska .NET w formie komponentów COM ... 962

Wprowadzenie do obsługi usług COM+ ... 964
Transakcje ... 966
Aktywacja kompilacji JIT ... 967
Tworzenie puli obiektów ... 968
Inne usługi ... 969
Podsumowanie ... 969

Rozdział 42. Debugowanie aplikacji z wykorzystaniem typów przestrzeni System.Diagnostics ... 971
Debugowanie w formie podstawowej ... 973
Debugowanie warunkowe ... 974
Śledzenie działania programu ... 977
Tworzenie asercji ... 979
Wykorzystywanie wbudowanych liczników wydajności ... 980
Implementacja zegarów ... 987
Tworzenie własnego licznika wydajności ... 988
Analiza wydajności na podstawie zebranych próbek ... 997
Podsumowanie ... 1005

Część X Wdrażanie kodu ... 1007

Rozdział 43. Złożenia i wersjonowanie ... 1009
Składniki złożenia ... 1010
Pliki manifestu ... 1011
Atrybuty ... 1011
Funkcjonalności złożeń ... 1013
Identyfikacja ... 1014
Zakres ... 1014
Wersjonowanie ... 1014
Zabezpieczenie ... 1014
Konfiguracja ... 1016
Sekcja Startup ... 1016
Sekcja Runtime ... 1017
Wdrażanie złożeń ... 1019
Podsumowanie ... 1019

Rozdział 44. Zabezpieczanie kodu ... 1021
Zabezpieczanie dostępu do kodu ... 1022
Dowody ... 1022
Uprawnienia ... 1023
Grupy kodowe ... 1023
Poziomy zabezpieczeń ... 1025
Żądania uprawnień ... 1026
Implementacja zasad bezpieczeństwa ... 1028
Zabezpieczanie za pomocą ról ... 1031
Narzędzia związane z systemami zabezpieczeń ... 1033
Podsumowanie ... 1033

Rozdział 45. Tworzenie pakietów instalacyjnych w środowisku Visual Studio 2008 ... 1035
Kreator tworzenia programów instalacyjnych w VS2008 ... 1036
Dodatkowe ustawienia konfiguracyjne programu instalacyjnego ... 1039
System plików ... 1039
Zapisy w rejestrze ... 1040
Typy plików ... 1040
Interfejs użytkownika ... 1040
Warunki instalacji ... 1042
Akcje niestandardowe ... 1042
Podsumowanie ... 1043

Rozdział 46. Wdrażanie aplikacji desktopowych ... 1045
Wdrożenie aplikacji z wykorzystaniem ClickOnce ... 1046
Konfigurowanie narzędzia ClickOnce ... 1049
Podsumowanie ... 1050

Rozdział 47. Rozpowszechnianie aplikacji sieciowych ... 1051
Anatomia aplikacji sieciowej ... 1052
Tworzenie serwera aplikacji sieciowych ... 1052
Tworzenie katalogu wirtualnego ... 1054
Wdrażanie aplikacji w serwerze ... 1055
Publikowanie aplikacji sieciowej bezpośrednio ze środowiska VS2008 ... 1056
Podsumowanie ... 1057

Część XI Dodatki ... 1059

Dodatek A Opcje kompilatora ... 1061
Opcje zaawansowane ... 1062
Opcje dla złożeń ... 1063

Dodatek B System pomocy w .NET Framework ... 1065
Książka ... 1066
Indeks ... 1066
Dokumentacja .NET Framework Class Library ... 1067
Mechanizmy wyszukiwania ... 1067
Ulubione strony internetowe ... 1068
Podsumowanie ... 1068

Skorowidz ... 1069

Rozdział 8.
Projektowanie obiektów

W świecie rzeczywistym, który różni się od *Second Life*[1] i innych światów wirtualnych, mamy codziennie do czynienia z wieloma rodzajami problemów. Zastanów się, w jaki sposób się je rozwiązuje. Na przykład czego potrzebowałbyś do zorganizowania specjalnego przyjęcia? Musiałbyś ustalić datę i godzinę spotkania, a także rodzaje rozrywek i zestawy potraw, zatrudnić pracowników i przygotować zaproszenia. Wszystkie te elementy musiałbyś skoordynować za pomocą planu działań.

Z punktu widzenia oprogramowania powyżej wspomniane elementy są obiektami, takimi jak klasa stworzona przez użytkownika lub struktura. Będą one następnie wzbogacone o możliwość przechowywania składników klasy, które definiują jej atrybuty oraz zachowanie. Atrybuty i sposób zachowania mogą zostać odwzorowane na elementy języka C#. Atrybut mógłby być polem lub właściwością, a zachowanie mogłoby być metodą lub zdarzeniem. Oto świat obiektowy — dzięki definiowaniu obiektów, które reprezentują fragmenty świata rzeczywistego, możesz tworzyć bardziej sensowne systemy.

W tym podrozdziale zostanie zaprezentowane, w jaki sposób tworzy się obiekty oraz definiuje ich składniki.

Elementy obiektu

Obiekt powinien być samodzielny i zaprojektowany do realizacji jednego celu. Wszystkie jego elementy powinny być zgodne i skutecznie ze sobą współpracować, aby wykonać założone zadanie. Oto szkielet prostej klasy:

```
class WebSite
{
    // konstruktory

    // destruktory

    // pola

    // metody

    // właściwości

    // indeksatory

    // zdarzenia

    // obiekty zagnieżdżone
}
```

[1] *Second Life* — częściowo płatny wirtualny świat 3D, udostępniony publicznie w 2003 roku przez firmę Linden Lab, mieszczącą się w San Francisco — *przyp. tłum.*

W powyższym przykładzie słowo kluczowe `class` oznacza klasę — typ referencyjny zdefiniowany przez użytkownika. Słowo `WebSite` jest nazwą klasy, a jej elementy zawarte są wewnątrz nawiasów klamrowych. Zamiast klasy mogłaby zostać zdefiniowana struktura, lecz wiązałoby się to z powstaniem typu wartościowego, zgodnie z tym, co przedstawiono w rozdziale 4., zatytułowanym „Typy referencyjne i wartościowe".

Często używane słowo: obiekt

Słowo *obiekt* jest używany na tak wiele sposobów, że nie dziwi to, iż wprowadza ono wielu ludzi w zakłopotanie. W informatyce obiekt jest przedmiotem, który reprezentuje pewną jednostkę w obszarze problemu do rozwiązania. Według terminologii .NET definicja obiektu zwana jest często typem. W programowaniu zorientowanym obiektowo obiekt jest konkretyzacją (instancją) typu. W języku C# słowo `object` oznacza typ bazowy dla wszystkich innych typów.

W tym rozdziale używam powyżej wspomnianej definicji informatycznej, aby zaprezentować, w jaki sposób w języku C# należy tworzyć obiekty. Możesz odwiedzić stronę *http://pl.wikipedia.org/wiki/Obiekt*, aby dowiedzieć się, ile znaczeń ma słowo *obiekt*.

W kolejnych podrozdziałach zapoznasz się szczegółowo ze wszystkimi składnikami obiektu. Są to między innymi pola, konstruktory, destruktory, metody, właściwości, indeksatory i zdarzenia.

Elementy statyczne i instancyjne

Każdy składnik obiektu może zostać przypisany do jednej z dwóch kategorii: elementu statycznego lub instancyjnego. Gdy tworzona jest kopia obiektu, powstaje jego nowy egzemplarz. W tym przypadku obiekt jest całkowicie niezależną jednostką, posiadającą swój własny zestaw atrybutów i sposób zachowania. Gdyby została stworzona druga instancja obiektu, miałaby ona niezależny zestaw danych w porównaniu z pierwszym egzemplarzem. W języku C# składniki obiektu należą domyślnie do instancji, chyba że zostaną zdefiniowane jako statyczne. Przykładem egzemplarza obiektu jest klasa `Customer`, która przechowuje odmienne informacje dla każdego klienta.

Jeśli użyjesz statycznego modyfikatora dla jakiegoś składnika obiektu, w danym momencie będzie istnieć tylko jedna jego kopia, bez względu na to, ile stworzono egzemplarzy obiektu. Elementy statyczne są przydatne podczas definiowania dostępu do pól statycznych (stanu obiektu). W bibliotece klas platformy .NET (FCL) można znaleźć wartościowe przykłady użycia metod statycznych, takie jak klasy `System.Math`, `System.IO.Path` i `System.IO.Directory`. Każda z nich ma elementy statyczne, które przetwarzają dane wejściowe i zwracają wartość. Ponieważ nie zależą one od żadnego stanu obiektu, wygodne jest zdefiniowanie ich jako składników statycznych, co zapobiega powstaniu narzutu podczas tworzenia instancji.

Pola

Pola zawierają zasadniczą część danych w klasie. Określają one stan obiektu. Są elementami klasy w przeciwieństwie do lokalnych zmiennych, które są definiowane wewnątrz metod i właściwości.

Pola mogą być inicjalizowane podczas deklaracji lub później, w zależności od stylu lub rodzaju wymagań. Każdy ze sposobów inicjalizacji ma swoje wady i zalety.

Na przykład ostrożną metodą może być zapewnienie, że wszystkie pola będą zawierały domyślne wartości, co wymusza ich inicjalizację podczas deklaracji lub zaraz po niej. Jest to zachowanie bezpieczne, a być może również zasadniczo wspomaga planowanie projektu poprzez żądanie zastanowienia się nad właściwościami definiowanych danych. Oto przykład deklaracji pola:

```
string siteName = "Zaawansowana strona o bezpieczeństwie komputerowym";
```

Deklaracja pola i jego inicjalizacja mogą zostać przeprowadzone w jednym wierszu. Nie jest to jednak koniecznie wymagane. Pola mogą być deklarowane w jednym wierszu, a inicjalizowane później, jak przedstawiono na poniższym przykładzie:

```
string url;

// gdzieś w dalszej części kodu
url = "http://www.comp_sec-mega_site.com";
```

W razie konieczności w jednym wierszu można deklarować wiele pól. Muszą one być od siebie oddzielane przecinkami. Takie wiersze mogą nawet zawierać deklarację jednego lub większej liczby pól, jak zaprezentowano w poniższym przykładzie:

```
string siteName, url, description = "Informacje o bezpieczeństwie
↪komputerowym";
```

Wszystkie trzy powyżej zadeklarowane pola są łańcuchami. Pole `description` zostało zainicjalizowane przy użyciu łańcucha literalnego. Inne pola są wciąż niezainicjalizowane, mogłyby jednak zostać zainicjalizowane w taki sam sposób, jak pole `description`.

Pola stałe

Gdy wartość pola jest znana wcześniej i nie zmienia się, możesz stworzyć stałą. Pole stałe gwarantuje, że jego wartość nie zostanie zmieniona podczas wykonania programu. Może być ono wielokrotnie odczytywane, nie można jednak do niego zapisywać ani w jakikolwiek sposób go modyfikować.

Pola stałe są wydajne. Ich wartości są znane podczas kompilacji. Pozwala to na zastosowanie pewnych optymalizacji niedostępnych dla innych rodzajów pól. Pola stałe są również z definicji statyczne. Oto przykład:

```
const string http = "http://";
```

W powyższym przykładzie przedstawiono deklarację stałej łańcuchowej zainicjalizowanej łańcuchem literalnym. Stałe mogą być inicjalizowane wartościami literalnymi. Jest to dobry wybór, ponieważ takie wartości się nie zmieniają. Zastanów się nad sposobem, w jaki czasami wprowadza się adresy internetowe w przeglądarce: użytkownik podaje fragment adresu, zakładając, że protokół internetowy będzie zgodny ze standardami WWW. Prostą metodą uzyskania tej funkcjonalności jest zadeklarowanie pola stałego definiującego protokół HTTP jako domyślny przedrostek dla wszystkich adresów sieciowych.

Stałe całkowite mogą być implementowane przy użyciu słowa kluczowego `const`, lecz często dużo wygodniejsze jest definiowanie ich w postaci typów wyliczeniowych. Ich użycie wymusza również implementację o ściślejszej kontroli typu. W rozdziale 6., zatytułowanym „Użycie tablic i typów wyliczeniowych", omówiono szczegółowo typy wyliczeniowe.

Pola readonly

Pola `readonly` są podobne do pól stałych, ponieważ nie mogą być modyfikowane po zainicjalizowaniu. Największą różnicą między tymi dwoma rodzajami pól jest moment inicjalizacji: stałe są inicjalizowane podczas kompilacji, natomiast pola `readonly` po uruchomieniu programu. Istnieją pewne powody, aby tak postępować, między innymi uniwersalność i udostępnianie większej funkcjonalności dla użytkowników.

Czasami wartość zmiennej pozostaje nieznana aż do uruchomienia programu. Może ona zależeć od różnych warunków i logiki programu. Pola `readonly` są inicjalizowane podczas tworzenia egzemplarza obiektu.

Pole `currentDate` w poniższym przykładzie zostaje zainicjalizowane wartością równą czasowi jego utworzenia:

```
readonly DateTime currentDate = DateTime.Now;
```

Ponieważ czas utworzenia obiektu jest wartością, której nie można ustalić podczas kompilacji programu, użycie modyfikatora `readonly` jest najodpowiedniejszą metodą rozwiązania tego przypadku.

Metody

Metody będą jednymi z najczęściej używanych przez Ciebie składników obiektów. Metoda w języku C# jest zbliżona do funkcji, procedur, podprogramów i tym podobnych elementów w innych językach programowania. O metodach C# można powiedzieć bardzo dużo — więcej dowiesz się o nich w rozdziale 10., zatytułowanym „Metody kodowania i operatory tworzone przez użytkownika". Teraz przedstawiam prosty przykład:

```
void MyMethod()
{
    // instrukcje programu
}
```

Powyższa metoda nie zwraca żadnej wartości, dlatego zwracany typ jest równy `void`. Metoda nazywa się `MyMethod`. Powinieneś jednakże nadawać metodom takie nazwy, które reprezentują ich działanie. Zaprezentowana metoda nie ma żadnych parametrów, lecz w dalszym ciągu do nazwy musisz dołączać nawiasy okrągłe. Blok kodu występujący wewnątrz nawiasów klamrowych jest treścią metody.

Właściwości

Właściwości w języku C# pozwalają na ochronę dostępu do stanu Twojego obiektu. Możesz używać ich jak pól, lecz działają one bardziej jak metody. W poniższych podrozdziałach zostanie zaprezentowane, w jaki sposób należy deklarować i używać właściwości. Poznasz również nową opcję wersji 3.0 języka C#, zwaną właściwością automatyczną (ang. *autoimplemented property*), oraz dowiesz się, jak w środowisku VS2008 należy używać gotowego fragmentu kodu dla właściwości.

Deklarowanie właściwości

Oto przykład prostej właściwości:

```
private string m_description;

public string Description
{
   get
   {
      return m_description;
   }

   set
   {
      m_description = value;
   }
}
```

Właściwość rozpoczyna się od modyfikatora dostępu `public`. Oznacza to, że kod znajdujący się poza aktualną klasą będzie miał dostęp do tej właściwości. Kolejnym elementem jest typ właściwości: `string`. Nazwą właściwości jest `Description`. Zawiera ona oba akcesory: `get` i `set`.

Akcesory `get` i `set` mogą zawierać dowolnie zdefiniowaną przez użytkownika logikę. Akcesor `get` zwraca wartość, natomiast akcesor `set` ustawia ją. Zwróć uwagę na obecność słowa kluczowego `value` w akcesorze `set`; przechowuje ono dowolną wartość, która została przypisana do właściwości.

Użycie właściwości

Oto przykład prezentujący użycie właściwości:

```
static void Main()
{
WebSite site = new WebSite();

site.Description = "świetna strona";
string desc = site.Description;
}
```

Załóżmy, że właściwość `Description` została zdefiniowana wewnątrz klasy `WebSite`. Możesz następnie stworzyć egzemplarz tej klasy — w powyższym kodzie jest to obiekt `site`. Poprzez niego masz dostęp do właściwości `Description`. Pamiętaj, że została ona zdefiniowana jako `public`, co oznacza, że jest widoczna poza klasą `WebSite`. Element `m_description` został jednak zdefiniowany jako `private`, dlatego nie jest on dostępny dla powyższego kodu.

Zwróć uwagę na to, w jaki sposób do elementu `site.Description` został przypisany łańcuch `"świetna strona"`. Gdy ta operacja ma miejsce, następuje wywołanie akcesora `set` dla właściwości `Description`. Oprócz tego słowo kluczowe `value` wewnątrz akcesora `set` przechowuje przypisany łańcuch `"świetna strona"`.

Następnie przyjrzyj się, jak do zmiennej `desc` przypisano wartość elementu `site.Description`. Podczas tej operacji wywołany zostaje akcesor `get` dla właściwości `Description`. Zwraca on wartość, która została przypisana do pola `m_description`. Zostaje ona z kolei przypisana do zmiennej `desc`.

Powyższy przykład prezentuje, w jaki sposób właściwość może używać pojedynczego pola jako swojej lokalnej pamięci pomocniczej. Wykonywana jest tylko operacja ustawiania lub pobierania wartości pola z tej pamięci — jest to powszechnie stosowany sposób działania. W rozdziale 9., zatytułowanym „Implementacja reguł zorientowanych obiektowo”, dowiesz się więcej o obiektowo zorientowanej zasadzie hermetyzacji, która wyjaśni, dlaczego powinno się używać właściwości w powyżej zaprezentowany sposób, zamiast bezpośrednio sięgać do pól klasy. W pierwszym rzędzie będziesz chciał zapewnić rozłączność obiektów oraz poprawić zarządzanie kodem — powyżej zalecane użycie właściwości jest w tym bardzo pomocne. W następnym podrozdziale przedstawiony zostanie łatwiejszy sposób rozwiązania prostego zadania polegającego jedynie na ustawianiu i pobieraniu wartości z lokalnej pamięci pomocniczej.

Właściwości automatyczne

Wykorzystanie właściwości obejmujących swoim zasięgiem jedynie pojedyncze pole jest tak powszechne, że w wersji 3.0 języka C# wprowadzono właściwości automatyczne. Oto przykład:

```
public int Rating { get; set; }
```

Właściwość `Rating` dotyczy pewnej wartości o typie `int`. Ponieważ wywołujący kod nie ma dostępu do lokalnej zmiennej pomocniczej, dlatego jej nazwa nie będzie mu do niczego potrzebna. Kompilator języka C# automatycznie utworzy pole o typie `int`.

Inną korzyścią z używania właściwości automatycznej jest eliminowanie pokusy pojawiającej się u programistów i polegającej na zajmowaniu się kodowaniem na poziomie lokalnej pamięci pomocniczej, zamiast obsługi samej właściwości. Na przykład w przypadku właściwości `Des ↪cription` kod znajdujący się w definiującej ją klasie ma dostęp do pola `m_description`. Może się jednak zdarzyć, że po jakimś czasie zmodyfikujesz kod implementujący tę właściwość, dzięki czemu będzie on zawierał reguły biznesowe przetwarzające wartość, która została do niej przypisana. Możliwe będzie, że inny kod w tej samej klasie nie zostanie odpowiednio zmieniony, aby używać zaimplementowanych reguł biznesowych, co spowoduje powstanie błędu w oprogramowaniu.

Gotowy fragment kodu dla właściwości w środowisku VS2008

W środowisku VS2008 istnieje kolejny przydatny snippet, którego możesz użyć podczas kodowania właściwości. Oto sposób jego wykorzystania:

1. Kliknij myszą wewnątrz klasy `WebSite`, aby umieścić tam kursor (jeśli Twój projekt nie zawiera klasy `WebSite`, stwórz ją).
2. Wprowadź z klawiatury ciąg znaków `pro` i naciśnij *Tab*, co spowoduje, że w edytorze pojawi się słowo `prop`. Dla właściwości istnieją również inne gotowe fragmenty kodów, lecz w tej chwili chcemy zająć się jedynie kodem `prop`.
3. Ponownie naciśnij klawisz *Tab*, aby uzyskać szkielet kodu dla właściwości. Podświetlone zostanie pole typu, które domyślnie równe jest `int`.
4. Wpisz `Web`, a następnie naciśnij *Tab*. W polu typu pojawi się słowo `WebSite`.
5. Naciśnij klawisz *Tab*, co spowoduje, że kursor przemieści się do następnego pola, którym będzie nazwa właściwości.
6. Wprowadź z klawiatury `BetaSite` i naciśnij klawisz *Enter*. Kursor przemieści się na koniec fragmentu kodu.

Stworzyliśmy właściwość automatyczną. Snippet `propg` tworzy właściwość z akcesorem `get`, a fragmenty kodu `propa` i `propdp` tworzą właściwości dołączone i zależne, których używa się w aplikacjach wykorzystujących systemy Windows Presentation Foundation i Windows Workflow. Zostaną one zaprezentowane w rozdziale 26., zatytułowanym „Tworzenie aplikacji Windows Presentation Foundation (WPF)", i rozdziale 28., zatytułowanym „Wykorzystanie ASP.NET AJAX w aplikacjach sieciowych".

Indeksatory

Indeksatory pozwalają na tworzenie obiektów, które mogą być następnie używane jak tablice. Można potraktować ich implementację jako połączenie tablicy, właściwości i metody.

Indeksatory zachowują się jak tablice, ponieważ używają składni wykorzystującej nawiasy kwadratowe, aby uzyskać dostęp do ich elementów. Zestaw klas platformy .NET używa indeksatorów w tym samym celu. Ich elementy są dostępne za pomocą indeksów.

Indeksatory zostały zaimplementowane jak właściwości, ponieważ wykorzystują akcesory `get` i `set` oraz ich składnię. Za pomocą akcesora `get` zwracają odpowiednią wartość wskazywaną przez indeks. Podobnie też przy użyciu akcesora `set` ustawiają wartość odpowiadającą danemu indeksowi.

Indeksatory wykorzystują również listę parametrów, podobnie jak ma to miejsce w przypadku metod. Lista parametrów jest zawarta w nawiasach okrągłych. Parametry mają zazwyczaj typ `int`, dzięki czemu klasa może udostępniać operacje podobne do tablic. Innymi użytecznymi typami są również `string` i `enum`. Oto przykład:

```
const int MinLinksSize = 0;
const int MaxLinksSize = 10;
string[] m_links = new string[MaxLinksSize];

public string this[int i]
{
   get
   {
      if (i >= MinLinksSize && i < MaxLinksSize)
      {
         return m_links[i];
      }
      return null;
   }

   set
   {
      if (i >= MinLinksSize && i < MaxLinksSize)
         {
            m_links[i] = value;
         }
   }
}

// kod w innej klasie

static void Main()
{
   WebSite site = new WebSite();
```

```
        site[0] = "http://www.mysite.com";
        string link = site[0];
    }
```

Indeksator z powyższego przykładu używa parametru całkowitoliczbowego. Akcesory `get` i `set` chronią przed próbami dostępu do elementów spoza zakresu.

Sposób użycia tego indeksatora jest bardzo podobny do wykorzystania tablicy. Pod koniec powyższego przykładu pojawia się instancja klasy `WebSite` będąca obiektem zawierającym indeksator. Podobnie jak ma to miejsce w przypadku właściwości, odczytywanie indeksatora wywołuje akcesora `get`, natomiast przypisywanie do indeksatora wywołuje akcesora `set`. Liczba znajdująca się w nawiasach jest przekazywana do parametru `i`, który wykorzystuje się w akcesorach; w powyższym przykładzie parametr `i` ma wartość `0`. Słowo kluczowe `value` przechowuje przypisaną wartość, która w tym przypadku jest równa `"http://www.mysite.com"`.

Gdzie mogą zostać użyte typy częściowe?

Typy częściowe, wprowadzone w wersji 2.0 języka C#, pozwalają na podział definicji pojedynczego typu na wiele części. Mimo że części mogą znajdować się w tym samym pliku, służą one zazwyczaj do umieszczenia definicji obiektu w wielu plikach. Podstawowym zadaniem typów częściowych jest wsparcie w oddzielaniu kodu wygenerowanego maszynowo od kodu tworzonego przez programistę. Na przykład w środowisku VS2008 kreatory projektów i elementów wykorzystujących ASP.NET i Windows Form tworzą szkielet klas umieszczonych w dwóch plikach. Zmniejsza to ilość kodu, z którym musisz bezpośrednio pracować, ponieważ Twoja część znajduje się w jednym pliku, natomiast w drugim zawarty jest segment wygenerowany maszynowo.

Składnia identyfikująca typ częściowy zawiera definicję klasy (lub struktury) z użytym modyfikatorem `partial`. W czasie kompilacji język C# identyfikuje wszystkie klasy zdefiniowane z tym samym modyfikatorem `partial` i łączy je w pojedynczy typ. W poniższym kodzie zaprezentowano składnię typów częściowych:

```
using System;

partial class Program
{
    static void Main()
    {
        m_someVar = 5;
    }
}

// kod umieszczony w innym pliku

using System;

partial class Program
```

```
{
   private static int m_someVar;
}
```

Powyższy kod znajduje się w dwóch różnych plikach. Drugi plik rozpoczyna się od drugiego użycia instrukcji `using System`. W prosty sposób zaprezentowałem tu deklarację typu częściowego, której obie części zawierają modyfikator `partial`. Zwróć uwagę na to, że pole `m_someVar` zostało zadeklarowane w jednej części, a użyte w metodzie `Main` znajdującej się w drugiej części. Podczas działania programu obie części tworzą jedną klasę, co nie stanowi żadnego problemu.

Zaprezentowana została podstawowa składnia typu częściowego; z jego działaniem będziesz mógł zapoznać się w rozdziale 25. „Tworzenie aplikacji w oparciu o formularze Windows Forms", rozdziale 26. „Tworzenie aplikacji Windows Presentation Foundation (WPF)" i rozdziale 27. „Tworzenie aplikacji sieciowych za pomocą ASP.NET".

Klasy statyczne

Standardowe klasy mogą zawierać elementy instancyjne i statyczne. Czasem jest jednak wymagane, aby klasa składała się jedynie z elementów statycznych. W tym przypadku możesz utworzyć klasę statyczną. Oto przykład:

```
public static class CustomMathLib
{
   public static double DoAdvancedCalculation(double param1, double param2)
   {
      return -1;
   }
}
```

Jak przedstawiono powyżej, w celu utworzenia klasy statycznej wystarczy użyć modyfikatora `static`. Wynika stąd, że wszystkie elementy klasy muszą również być statyczne.

Klasa System.Object

Jak dowiedziałeś się już w rozdziale 4., wszystkie typy pochodzą z klasy `System.Object`. Ze względu na ten związek dziedziczenia wszystkie obiekty zawierają w sobie także elementy tej klasy. W tym podrozdziale zostanie zaprezentowane, czym są te elementy i jak należy ich używać.

Sprawdzanie typu obiektu

W rozdziale 2., zatytułowanym „Wprowadzenie do języka C# i środowiska Visual Studio 2008", poznałeś operator `typeof`. Przyjrzałeś się jego praktycznemu wykorzystaniu w rozdziale 6. podczas pracy z klasą `Enum`. Cechą operatora `typeof` jest to, że wymaga on znajomości typu użytego

parametru. Czasem jednakże będziesz miał do czynienia z obiektem, którego typu być może nie będziesz znać. Na przykład podczas użycia ogólnej metody z parametrem typu obiektowego będziesz chciał sprawdzić, czy aktualnie używany typ jest prawidłowy. Z taką sytuacją spotkasz się w rozdziale 9. podczas własnej implementacji metody `Equals`. Aby uzyskać typ egzemplarza obiektu, możesz wywołać jego metodę `GetType`:

```
Type siteType = site.GetType();
```

W powyższym przykładzie wywołano metodę `GetType` dla instancji `WebSite`. Być może jest to klasa pochodząca z `WebSite`, dla której należy poznać dokładny typ.

Porównywanie referencji

Inną metodą klasy `System.Object` jest `ReferenceEquals`, która działa z obiektami o typie referencyjnym. Dzięki niej będziesz mógł się dowiedzieć, czy dwie zmienne zawierają referencję do tego samego obiektu. Oto przykład:

```
WebSite site2 = site;
bool isSameObject = object.ReferenceEquals(site, site2);
```

Ponieważ w powyższym kodzie przypisano zmienną `site` do `site2`, ich referencje będą takie same, dlatego wywołanie metody `ReferenceEquals` zwróci `true`. Mógłbyś mieć dwa odmienne obiekty o tych samych wartościach, lecz metoda `ReferenceEquals` zwróciłaby w tym przypadku `false`, ponieważ referencje nie byłyby sobie równe.

Sprawdzanie równości

Przeznaczeniem metody `Equals` jest sprawdzanie, czy wartości są sobie równe. Dla obiektów o typie wartościowym metoda ta sprawdza automatycznie wszystkie ich elementy. W przypadku obiektów o typie referencyjnym metoda `Equals` wywołuje jednakże `ReferenceEquals`, która zwraca informację o równości referencji. W rozdziale 9. zostanie przedstawione, w jaki sposób do swoich obiektów możesz dodać metodę `Equals`, aby zdefiniować równość wartości. Oto kilka przykładów prezentujących użycie instancyjnych i statycznych metod `Equals` z klasy `System.`
↪`Object`:

```
WebSite site3 = new WebSite();
WebSite site4 = new WebSite();
site3.Description = "Informacje o C#";
site4.Description = site3.Description;

bool isSiteEqual = site3.Equals(site4);
isSiteEqual = object.Equals(site3, site4);
```

W powyższym przykładzie właściwość `site4.Description` została ustawiona na wartość właściwości `site3.Description`, dzięki czemu oba obiekty uzyskały taką samą wartość. Nie ma to jednakże znaczenia, ponieważ oba wywołania metody `Equals` zwracają `false`. Dzieje się tak, gdyż zmienne odnoszą się do dwóch różnych obiektów. W rozdziale 9. pokażę, jak można to naprawić.

Uzyskiwanie wartości mieszających

Często wykorzystywaną operacją podczas programowania w języku C# jest praca z tablicami mieszającymi (ang. *hash tables*), które przechowują obiekty w oparciu o ich niezmienne klucze. Tablice te są również nazywane w innych językach asocjacjami, tablicami asocjacyjnymi lub słownikami. Aby wspomóc tworzenie kluczy, klasa `System.Object` zawiera metodę `GetHashCode` przedstawioną poniżej:

```
int hashCode = site3.GetHashCode();
```

W zasadzie nie będziesz wywoływać metody `GetHashCode` w powyżej przedstawiony sposób, chyba że zamierzasz zaimplementować swoją własną tablicę mieszającą. Zostanie ona wywołana podczas użycia klasy `Hashtable` lub ogólnej klasy `Dictionary`, co zaprezentowano w rozdziale 17., zatytułowanym „Parametryzowanie typów poprzez szablony klas i tworzenie iteratorów". Domyślna implementacja metody `GetHashCode` z klasy `System.Object` nie gwarantuje zwracania unikalnej wartości, jednak w rozdziale 9. dowiesz się, jak można zdefiniować własną metodę tego typu.

Klonowanie obiektów

Klasa `System.Object` zawiera także metodę zwaną `MemberwiseClone` tworzącą kopie obiektów. Oto przykład:

```
// element klasy WebSite
public WebSite GetCopy()
{
   return MemberwiseClone() as WebSite;
}
```

Ponieważ metoda `MemberwiseClone` została zdefiniowana z modyfikatorem `protected`, musiałem wywołać ją z wnętrza klasy `WebSite`. Oznacza to, że dostęp do tej metody mają tylko klasy pochodne. Więcej o użyciu modyfikatora `protected` i innych sposobach definiowania dostępu dowiesz się w rozdziale 9. Oto kod, który wykorzystuje tę metodę:

```
WebSite beta = new WebSite();
WebSite site5 = new WebSite();
site5.BetaSite = beta;

WebSite site6 = site5.GetCopy();

bool areSitesEqual = ReferenceEquals(site5, site6);
bool areBetasEqual = ReferenceEquals(site5.BetaSite, site6.BetaSite);

Console.WriteLine("Równość obiektów site: {0}, równość właściwości beta: {1}",
↪areSitesEqual, areBetasEqual);
```

Zwróć uwagę na to, że właściwość `BetaSite` dla obiektu `WebSite` została ustawiona na wartość `beta`. Oznacza to, że obiekt `WebSite` zawiera pole z referencją do innej instancji klasy `WebSite`. Podczas wywołania metody `GetCopy`, która z kolei wywołuje `MemberwiseClone`, do zmiennej `site6` przypisany zostaje nowy obiekt będący kopią `site5`.

Oto coś, co może Cię zaskoczyć: metoda `MemberwiseClone` przeprowadza tylko płytkie kopiowanie. Płytkie kopiowanie wykonuje kopię obiektów wyłącznie na pierwszym poziomie grafu obiektowego. Obie zmienne `site5` i `site6` znajdują się właśnie na tym poziomie grafu obiektowego. Instancja `beta` klasy `Website` znajduje się jednakże w obiekcie `site5` na drugim poziomie grafu obiektowego. Oznacza to, że jedynym elementem, który zostanie skopiowany podczas wywołania metody `MemberwiseClone`, będzie referencja przechowywana we właściwości `Beta`
`↪Site` obiektu `site5`. Tak więc mimo że `site5` i `site6` są odmiennymi obiektami, które naprawdę zostały skopiowane, jednakże odwołują się do tego samego obiektu przypisanego do `BetaSite`. Poniżej przedstawiony wynik działania kodu potwierdza tę uwagę:

```
Równość obiektów site: False, równość właściwości beta: True
```

Zapamiętaj, że metoda `MemberwiseClone` wykonuje płytkie kopiowanie. W celu przeprowadzenia głębokiego kopiowania będziesz musiał zaimplementować własną metodę.

Używanie obiektów jako łańcuchów

Być może do tej pory zwróciłeś uwagę na jeden fakt: metodę `ToString` możesz wywołać dla dowolnego elementu. Metoda `Console.WriteLine` wywołuje domyślnie `ToString` dla każdego parametru, który zostanie do niej dostarczony. Dzieje się tak, ponieważ `ToString` jest elementem klasy `System.Object`. Oto kilka przykładów:

```
string siteStr = site6.ToString();
string fiveStr = 5.ToString();

Console.WriteLine("site6: {0}, pięć: {1}", siteStr, fiveStr);
```

Do klasy `WebSite` nie dołączyliśmy metody `ToString`, lecz mimo to może ona zostać wywołana. Możesz również wywołać ją dla wartości literalnej, jak zaprezentowano w powyższym przykładzie w przypadku liczby `5`. Oto wynik działania kodu:

```
site6: Rozdzial_08.WebSite, pięć: 5
```

Wynik działania dla liczby `5` nie jest zaskakujący, lecz zauważ, co uzyskano dla zmiennej `site6`. `Rozdzial_08` jest przestrzenią nazw programu, której używam w tym rozdziale, a `WebSite` jest nazwą klasy. Jest to pełna kwalifikowana nazwa typu, którą domyślnie otrzymasz za pomocą metody `ToString`.

Domyślna wartość ToString podczas debugowania programu

Może się zdarzyć, że będziesz debugował swój program lub wydrukujesz wyniki zwracane przez metodę `ToString`, i zobaczysz pełną kwalifikowaną nazwę typu. Może to oznaczać, że obserwujesz lub wykorzystujesz niewłaściwy obiekt.

Na przykład w środowisku ASP.NET możesz do list przypisać obiekty `ListItem`, które zawierają właściwości `Text` i `Value`. Jednakże przypadkiem mógłbyś wywołać dla obiektu `ListItem` metodę `ToString`, zamiast właściwości `Text`, co było Twoim pierwotnym zamiarem.

Podsumowanie

Zapoznałeś się już z elementami klas i wiesz już, z czego składa się obiekt. Właściwości są przydatne podczas opisywania stanu obiektów. Możesz również używać obiektów jak tablic dzięki zaimplementowaniu indeksatorów.

Środowisko VS2008 wykorzystuje typy częściowe dla ułatwienia pracy z technologiami ASP.NET i Windows Forms dzięki umieszczaniu klas w różnych plikach.

Klasa `System.Object`, która — jak już wiesz — jest podstawową klasą dla wszystkich typów języka C#, zawiera elementy mogące zostać wykorzystane przez Ciebie. Możesz porównywać obiekty pod względem równości referencji i wartości. W następnym rozdziale poznasz różne koncepcje związane z programowaniem obiektowym, między innymi polimorfizm. Możesz go wykorzystać, aby przesłonić `Equals` i inne metody pochodzące z `System.Object`, a przez to uzyskać bardziej funkcjonalne klasy.