

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

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

Microsoft Visual C# 2005. Księga eksperta

Autor: Kevin Hoffman

Tłumaczenie: Marcin Winiarski (wstęp, rozdz. 1-23, 30-35, 38-41), Tomasz Grochowski (rozdz. 24-27, 29, 36, 37), Dariusz Matyszko (rozdz. 28)

ISBN: 978-83-246-0664-1

Tytuł oryginału: [Microsoft Visual C# 2005 Unleashed](#)

Format: B5, stron: około 650



Kompletny przewodnik po języku C#

- Poznaj składnię języka C# i możliwości platformy .NET
- Wykorzystaj formularze do tworzenia efektownych interfejsów użytkownika
- Pisz bezpieczne aplikacje bazodanowe i sieciowe

Na platformie .NET można programować w wielu językach, jednak najważniejszym z nich jest C#, który został zaprojektowany specjalnie w tym celu; to język ułatwiający wygodne korzystanie z niesamowitych możliwości .NET Framework. Za pomocą C# możesz bez żmudnej nauki skomplikowanej składni błyskawicznie tworzyć efektowne programy dla systemu Windows, witryny internetowe, usługi sieciowe czy aplikacje bazodanowe. Szerokie zastosowania i duże możliwości połączone z łatwością nauki sprawiają, że język C# to doskonały wybór zarówno dla początkujących, jak i doświadczonych programistów.

Książka „Microsoft Visual C# 2005. Księga eksperta” to wyczerpujący przegląd najnowszej i najbardziej użytecznej wersji języka C#. Czytając ją, poznasz podstawowe elementy i składnię języka, a także używane w nim struktury danych i techniki programowania obiektowego. Dowiesz się, czym jest platforma .NET, jak działa, jakie ma możliwości oraz – co najważniejsze – jak wykorzystać ją do pisania własnych programów. Nauczysz się obsługiwać bazy danych przy użyciu technologii ADO.NET, tworzyć różnorodne aplikacje sieciowe, a także stosować formularze Windows do projektowania programów dla tego systemu. Zrozumiesz techniki tworzenia aplikacji rozproszonych za pomocą technologii remoting.

- Składnia języka C#
- Działanie platformy .NET
- Używanie formularzy do tworzenia efektownych interfejsów
- Praca z bazami danych w ADO.NET 2.0
- Tworzenie aplikacji sieciowych w ASP.NET 2.0
- Pisanie i konsumowanie usług sieciowych
- Zabezpieczanie programów i danych
- Aplikacje rozproszone i technologia remoting
- Instalowanie aplikacji za pomocą technologii ClickOnce
- Programowanie wielowątkowe

**Jeśli chcesz nadążyć za najnowszą technologią
– ta książka jest dla Ciebie**

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



Spis treści

O autorze	17
Wstęp	19
Część I Podstawy C# 2.0	23
Rozdział 1. Wprowadzenie do C# 2.0	25
Czym jest .NET Framework?	25
Ewolucja .NET	25
Wspólne środowisko uruchomieniowe	26
Wspólny system typów	27
Wyrzucanie śmieci: programowanie w środowisku z mechanizmem przywracania pamięci	27
Praca ze zmiennymi w C# 2.0	28
Wspólne typy w .NET	28
Skrócone nazwy typów	29
Typy wartościowe a typy referencyjne	29
Podstawowa składnia C#	30
Bloki kodu	30
Klasyczny przykład „Witaj Świecie”	31
Co można zrobić z użyciem C#?	32
Podsumowanie	33
Rozdział 2. Wyrażenia i struktury sterujące	35
Rozgałęzianie i logika warunkowa	35
Wprowadzenie do wyrażeń logicznych	35
Zastosowanie podstawowych instrukcji warunkowych	37
Zastosowanie zaawansowanych instrukcji warunkowych	39
Zapętlanie i powtarzalność	39
Zastosowanie pętli for	40
Zastosowanie pętli while	40
Zastosowanie pętli do	41
Podsumowanie	41
Rozdział 3. Ciągi znaków i wyrażenia regularne	43
Praca z ciągami znaków	43
Wprowadzenie do ciągów znaków w .NET	43
Formatowanie ciągów znaków	44

Manipulowanie ciągami znaków oraz ich porównywanie	47
Wprowadzenie do klasy StringBuilder	49
Praca z wyrażeniami regularnymi	49
Walidacja wprowadzanych danych	50
Wydrebnianie wprowadzanych danych	50
Podsumowanie	51
Rozdział 4. Tablice i kolekcje	53
Praca z tablicami	53
Deklaracja i inicjalizacja tablic	53
Stosowanie tablic jednowymiarowych	54
Stosowanie tablic wielowymiarowych	56
Zastosowanie tablic postrzępionych	59
Praca z kolekcjami	61
Porównanie tablic i kolekcji	62
Zastosowanie klasy ArrayList	62
Zastosowanie klasy Hashtable	64
Zastosowanie klasy Queue	65
Zastosowanie klasy Stack	67
Zastosowanie klasy SortedList	67
Podsumowanie	68
Rozdział 5. Programowanie obiektowe w C#	69
Projektowanie obiektowe	69
Wprowadzenie do projektowania obiektowego	69
Projektowanie klas	70
Projektowanie interfejsów	71
Programowanie obiektowe	72
Tworzenie prostej klasy	72
Obsługa widoczności składowych	74
Zastosowanie dziedziczenia	75
Wprowadzenie do polimorfizmu	77
Implementacja interfejsów	78
Podsumowanie	81
Rozdział 6. Wprowadzenie do typów ogólnych	83
Przegląd typów ogólnych	83
Korzyści z zastosowania typów ogólnych	83
Wprowadzenie do parametrów typu	84
Ograniczenia dla parametrów typu	85
Budowa typów ogólnych	87
Tworzenie klas ogólnych	87
Tworzenie metod ogólnych	87
Tworzenie interfejsów ogólnych	89
Zastosowanie kolekcji ogólnych	90
Zastosowanie klasy Dictionary	90
Zastosowanie klasy List	90
Zastosowanie klasy Queue	91
Zastosowanie klasy Stack	91
Podsumowanie	92

Część II Podstawowe wiadomości o .NET Framework 2.0	93
Rozdział 7. Wejście/wyjście i trwałość	95
Wprowadzenie do strumieni	95
Zastosowanie strumieni pamięciowych	96
Wprowadzenie do podstawowych operacji wejścia/wyjścia na plikach	99
Tworzenie plików i dopisywanie do nich	99
Odczyt z istniejących plików	100
Operacje na katalogach i systemie plików	101
Zastosowanie asynchronicznych operacji wejścia/wyjścia na plikach	103
Praca z pamięcią izolowaną	104
Podsumowanie	106
Rozdział 8. Praca z XML	107
Odczyt i zapis dokumentów XML	107
Zapytania do XML z użyciem XPath	111
Przekształcanie dokumentów z użyciem XSLT	113
Walidacja dokumentów z użyciem XSD	115
Podsumowanie	117
Rozdział 9. Zdarzenia i delegaty	119
Wprowadzenie do delegatów	119
Kowariancja i kontrawariancja	121
Zastosowanie metod anonimowych	122
Tworzenie delegatów zbiorowych	124
Wprowadzenie do zdarzeń	126
Zaawansowane programowanie oparte na zdarzeniach	128
Podsumowanie	133
Rozdział 10. Programowanie wielowątkowe	135
Podstawy programowania wielowątkowego	135
Pierwsza aplikacja wielowątkowa	138
Tworzenie i uruchamianie wątku	138
Zakończenie wątku	139
Zawieszanie wątku	141
Usypianie wątku	141
Dołączanie do wątku	141
Synchronizacja wątków i ich rywalizacja o zasoby	142
Zastosowanie słowa kluczowego lock	143
Zastosowanie klasy Mutex	144
Zastosowanie klasy Monitor	146
Zastosowanie klasy Interlocked	147
Zastosowanie klasy ReaderWriterLock	147
Praca z ręcznym i automatycznym resetowaniem zdarzeń	149
Zastosowanie klasy ThreadPool	151
Podsumowanie	152
Rozdział 11. Podstawowe zasady refleksji	153
Wprowadzenie do refleksji	153
Praca z informacjami o metodach	155
Praca z informacjami o składowych	157

Badanie zdarzeń	160
Tworzenie i badanie atrybutów kodu użytkownika	161
Podsumowanie	163
Rozdział 12. Zestawy i domeny aplikacji	165
Wprowadzenie do zestawów	165
Zestawy — co się za tym kryje?	166
Manifest zestawu	167
Budowa i zastosowanie zestawów	168
Przechowywanie i wyszukiwanie zasobów zestawów	170
Lokalizacja i zestawy satelickie	172
Wprowadzenie do domen aplikacji	174
Programowanie z użyciem klasy AppDomain	176
Podsumowanie	177
Rozdział 13. Współoperatywność Windows i COM	179
Wprowadzenie do zagadnienia współoperatywności w C#	179
Zastosowanie obiektów COM z poziomu platformy .NET	180
Zastosowanie zestawów PIA (Primary InterOp Assembly)	181
Zastosowanie klas .NET z poziomu COM	183
Uzyskiwanie dostępu do kodu umieszczonego w niezarządzanych bibliotekach DLL	186
Podsumowanie	189
Rozdział 14. Zabezpieczenia dostępu kodu	191
Wprowadzenie do zabezpieczeń dostępu kodu	191
Zastosowanie polityki bezpieczeństwa i administrowanie nią	192
Administrowanie zabezpieczeniami dostępu kodu	194
Zastosowanie zabezpieczeń programowych	195
Programowe wymuszanie tożsamości	198
Zastosowanie zabezpieczeń deklaratywnych	200
Podsumowanie	201
Rozdział 15. Kryptografia i ochrona danych	203
Wprowadzenie do kryptografii	203
Szyfrowanie z kluczem tajnym	204
Szyfrowanie z kluczem publicznym	204
Tworzenie skrótów danych	205
Podpisy cyfrowe	206
Zastosowanie szyfrowania z kluczem tajnym	206
Zastosowanie szyfrowania z kluczem publicznym	208
Praca ze skrótami i podpisami cyfrowymi	210
Zastosowanie API zabezpieczania danych (DPAPI)	212
Podsumowanie	215
Rozdział 16. Optymalizacja kodu .NET 2.0	217
Pojęcia pakowania oraz rozpakowywania	217
Zastosowanie odpowiednich technik manipulowania ciągami znaków	219
Konstrukcja efektywnej pętli	220
Skracanie czasu rozruchu aplikacji	221
Profilowanie kodu z zastosowaniem narzędzia Performance Wizard	222
Podsumowanie	226

Część III Dostęp do danych z .NET 2.0	227
Rozdział 17. Podstawy ADO.NET	229
Wprowadzenie do ADO.NET	229
Nawiązywanie połączenia	230
Tworzenie łańcuchów połączeń	230
Zastosowanie klas DBConnection	231
Komunikacja ze źródłem danych	233
Wykonywanie poleceń	233
Zastosowanie obiektów typu DataReader	236
Zastosowanie mechanizmu Schema Discovery	238
Praca z danymi	239
Wprowadzenie do obiektów typu DataSet	240
Zastosowanie klasy DataAdapter	242
Podsumowanie	245
Rozdział 18. Zaawansowane techniki ADO.NET	247
Praca z nową, udoskonaloną klasą DataTable	247
Wczytywanie i zapisywanie zawartości obiektów DataTable z użyciem XML	248
Zastosowanie nowej klasy DataTableReader	250
Asynchroniczny dostęp do danych	250
Uaktualnianie danych w trybie wsadowym	252
Zastosowanie nowej przestrzeni nazw System.Transactions	254
Jawne zastosowanie transakcji	255
Niejawne zastosowanie transakcji	256
Podsumowanie	259
Rozdział 19. Praca z dostawcami danych ADO.NET	261
Wprowadzenie do źródeł danych ADO.NET	261
Zastosowanie fabryk dostawców	262
Pobieranie listy zainstalowanych fabryk dostawców	262
Zastosowanie fabryki dostawcy do nawiązania połączenia	263
Praca z łańcuchami połączenia	265
Wyliczanie źródeł danych	266
Uzyskiwanie dodatkowych informacji o dostawcy	267
Zastosowanie metody RetrieveStatistics()	267
Uzyskiwanie informacji o schemacie od dostawcy danych	268
Tworzenie dostawców danych ADO.NET użytkownika	269
Podsumowanie	270
Rozdział 20. Klasy DataSet ze ścisłą kontrolą typów	271
Wprowadzenie do Klasy DataSet z kontrolą typów	271
Zastosowanie schematów XSD w tworzeniu obiektów DataSet z kontrolą typów	272
Zastosowanie Designera przy tworzeniu obiektów DataSet z kontrolą typów	273
Programowanie z użyciem obiektu DataSet z kontrolą typów	274
Łączenie klas DataSet z kontrolą typów i danych dynamicznych	275
Ręczne wypełnianie obiektu DataSet z użyciem komponentu DataAdapter	276
Wypełnianie klas DataSet z kontrolą typów z użyciem obiektów typu DataAdapter	276
Dodawanie zapytań użytkownika do obiektów DataSet z kontrolą typów	280

Adnotacje do klasy DataSet z kontrolą typów	281
Rozszerzanie obiektu DataSet z kontrolą typów za pomocą klas częściowych	284
Podsumowanie	285

Rozdział 21. Programowanie z SQL Server 2005 287

Zapoznanie się z hostem SQL Server 2005 CLR	287
Budowa procedur składowanych w C#	288
Budowa funkcji definiowanych przez użytkownika w C#	291
Tworzenie typów definiowanych przez użytkownika w C#	292
Praca z nową biblioteką SQL po stronie serwera	297
Zastosowanie MARS (Multiple Active Result Sets)	300
Podsumowanie	302

Część IV Tworzenie aplikacji Web ASP.NET 2.0 305

Rozdział 22. Wprowadzenie do ASP.NET i formularzy Web 307

Wprowadzenie do ASP.NET 2.0	307
Hierarchia stron i kontrolek	308
Wprowadzenie do pojęcia kompilacji w ASP.NET 2.0	308
Bezpieczeństwo	309
Zarządzanie stanem	309
System konfiguracji aplikacji Web	310
Cykl życia strony ASP.NET	310
Etap cyklu życia strony ASP.NET	310
Zdarzenia zachodzące w cyklu życia strony ASP.NET	312
Przegląd kontrolek dostępnych w ASP.NET	315
Tworzenie i debugowanie aplikacji ASP.NET	315
Budowa i instalacja aplikacji ASP.NET	318
Debugowanie aplikacji ASP.NET	320
Obsługa zdarzeń i formularzy zwrotnych	320
Budowa dynamicznych stron interaktywnych z użyciem wywołań zwrotnych klienta	322
Podsumowanie	330

Rozdział 23. Zarządzanie stanem w ASP.NET 2.0 331

Praca ze stanem aplikacji	331
Praca ze stanem sesji	334
Zastosowanie domyślnego wewnętrznego dostawcy stanu	334
Zastosowanie dostawcy stanu ASP.NET State Server	338
Zastosowanie serwera SQL Server jako dostawcy stanu sesji	342
Obsługa zdarzeń stanu sesji	344
Praca ze stanem widoku	346
Zarządzanie stanem w farmach Web	348
Zastosowanie stanu aplikacji w farmach serwerów Web	348
Zastosowanie stanu sesji w farmach serwerów Web	349
Zastosowanie stanu widoku w farmach serwerów Web	349
Podsumowanie	351

Rozdział 24. Zastosowanie mechanizmu stron wzorcowych (Master Pages)	353
Świat przed pojawieniem się stron wzorcowych	353
Potrzeba spójnego interfejsu GUI	354
Tworzenie spójnego GUI w ASP.NET 1.1	354
Wprowadzenie do zagadnienia stron wzorcowych	355
Szablony stron i strony zawartości	355
Tworzymy pierwszy szablon strony	356
Tworzymy pierwszą stronę zawartości	357
Korzystanie z domyślnego szablonu	358
Strony wzorcowe „od kuchni”	359
Zaawansowane techniki stron wzorcowych	359
Zagnieżdżone szablony stron	360
Właściwość Master	361
Szablony stron z silną kontrolą typów	362
Obsługa względnych ścieżek	362
Podsumowanie	363
Rozdział 25. Personalizacja i dostosowywanie w ASP.NET	365
Tworzenie interfejsu użytkownika z kompozycjami i skórkami	365
Praca z profilami użytkownika w ASP.NET	369
Konfiguracja usług aplikacji	370
Konfiguracja dostawcy profili	371
Korzystanie z profili ASP.NET	372
Dostosowywanie dla użytkownika z kompozycjami i profilami	376
Podsumowanie	378
Rozdział 26. Wprowadzenie do kontrolki Web Parts	381
Podstawy kontrolki Web Parts	381
Korzystanie z dostawcy personalizacji	383
Tworzenie pierwszej strony Web Part	386
Tworzenie kontrolki Web Parts	394
Tworzenie powiązanych kontrolki Web Parts	395
Podsumowanie	399
Rozdział 27. Tworzenie bogatych aplikacji Web sterowanych przepływem danych	401
Wprowadzenie do zagadnienia wiązania z danymi w ASP.NET	401
Model źródła danych	402
Tworzenie źródeł danych	402
Hierarchia kontrolki korzystających z danych	405
Korzystanie z kontrolki dostępu do danych	406
Korzystanie z kontrolki GridView	406
Korzystanie z kontrolki DetailsView	407
Korzystanie z kontrolki FormView	407
Korzystanie z kontrolki TreeView	409
Zaawansowane techniki wiązania z danymi	410
Tworzenie obiektowego źródła danych i korzystanie z niego	411
Podsumowanie	412

Rozdział 28. Zabezpieczanie aplikacji ASP.NET 413

Bezpieczeństwo poprzez uwierzytelnianie	413
Uwierzytelnianie systemu Windows	414
Uwierzytelnianie w oparciu o usługę Microsoft Passport	415
Uwierzytelnianie oparte na formularzach	416
Zarządzanie kontami użytkowników (Membership API)	419
Bezpieczeństwo poprzez autoryzację	422
Autoryzacja z wykorzystaniem ról użytkowników	422
Kontrolki zabezpieczeń ASP.NET	425
Login	425
LoginName	426
LoginStatus	426
LoginView	427
PasswordRecovery	428
ChangePassword	429
CreateUserWizard	429
Zaawansowane bezpieczeństwo ASP.NET	430
Chronione ustawienia konfiguracyjne	430
Podsumowanie	432

Rozdział 29. Tworzenie własnych dostawców ASP.NET 433

Dostawca członkostwa	433
Wprowadzenie do bazowej klasy MembershipProvider	434
Implementacja schematu członkostwa	434
Przygotowanie własnego dostawcy członkostwa	436
Konfiguracja i instalacja dostawcy członkostwa	443
Dostawca ról	444
Wprowadzenie do bazowej klasy RoleProvider	444
Implementacja schematu ról	445
Przygotowanie własnego dostawcy ról	446
Konfiguracja i instalacja dostawcy ról	450
Dostawca profili ProfileProvider	450
Wprowadzenie do bazowej klasy ProfileProvider	450
Implementacja schematu profili	451
Tworzenie własnego dostawcy profili	452
Konfiguracja i instalacja dostawcy profili	456
Dodatkowi dostawcy	456
Wprowadzenie do dostawcy SessionState	457
Wprowadzenie do dostawcy SiteMap	457
Podsumowanie	458

Rozdział 30. Tworzenie kontrolki ASP.NET 461

Budowa kontrolki użytkownika	461
Tworzenie kontrolki serwera	464
Zarządzanie stanem z poziomu kontrolki serwera	468
Podsumowanie	473

Rozdział 31. Zarządzanie i monitorowanie ASP.NET	475
Prezentacja nowego systemu monitorowania sprawności	475
Zastosowanie systemu monitorowania sprawności	477
Tworzenie własnych zdarzeń	479
Tworzenie własnych dostawców zdarzeń	483
Zastosowanie liczników wydajności ASP.NET	484
Podsumowanie	487

Część V Usługi Web

Rozdział 32. Udostępnianie funkcjonalności za pomocą usług Web	491
Wprowadzenie do usług Web	491
Jak działają usługi Web	492
Tworzenie prostej usługi typu „Witaj Świecie”	493
Tworzenie usług transakcyjnych	499
Zarządzanie stanem w usługach Web	500
Podsumowanie	502

Rozdział 33. Zaawansowane programowanie usług Web	503
Projektowanie architektur zorientowanych na usługi (SOA)	503
Luźne sprzężenie a zależności	504
SOA jako sposób bycia	504
Zastosowanie Web Service Discovery	507
Zastosowanie niestandardowych nagłówków SOAP	509
Programowanie bezpiecznych usług Web	512
Sprzężanie formularzy Windows z usługami Web	514
Podsumowanie	516

Część VI Programowanie aplikacji Windows Forms 2.0

Rozdział 34. Wprowadzenie do Windows Forms 2.0	519
Podstawy Windows Forms	519
Windows Forms kontra Web Forms	520
Tworzenie aplikacji w technologii Windows Forms	522
Zastosowanie projektanta Windows Forms	523
Zastosowanie okna Document Outline	524
Wyrównywanie kontrolki za pomocą SnapLines	524
Tworzenie formularzy o zmiennych rozmiarach	526
Elementy poprawnego projektowania interfejsów użytkownika	528
Kolory w projekcie	528
Świadomość rozmiarów w projekcie	528
Złożoność projektu	529
Świadomość ilości kliknięć w projekcie	529
Intuicyjność projektu	529
Podsumowanie	529

Rozdział 35. Biblioteka kontrolki Windows Forms 531

Okno narzędziowe wspólnych kontrolki	531
Kontrolka Button	532
Kontrolka CheckBox	532
Kontrolka CheckedListBox	532
Kontrolka ComboBox	532
Kontrolka DateTimePicker	533
Kontrolka Label	533
Kontrolka LinkLabel	533
Kontrolka ListBox	534
Kontrolka ListView	534
Kontrolka MaskedTextBox	535
Kontrolka MonthCalendar	535
Kontrolka NotifyIcon	535
Kontrolka NumericUpDown	536
Kontrolka PictureBox	536
Kontrolka ProgressBar	536
Kontrolka RadioButton	537
Kontrolka TextBox	537
Kontrolka RichTextBox	537
Kontrolka ToolTip	537
Kontrolka TreeView	538
Kontrolka WebBrowser	538
Kontenery	539
Kontrolka FlowLayoutPanel	539
Kontrolka GroupBox	539
Kontrolka Panel	540
Kontrolka SplitContainer	540
Kontrolka TabControl	540
Kontrolka TableLayoutPanel	540
Menu i paski narzędziowe	541
Kontrolka ContextMenuStrip	541
Kontrolka MenuStrip	541
Kontrolka StatusStrip	541
Kontrolka ToolStrip	542
Kontrolka ToolStripContainer	542
Kontrolki związane z danymi	542
Kontrolka DataSet	542
Kontrolka DataGridView	543
Kontrolka BindingSource	543
Kontrolka BindingNavigator	543
Kontrolka ReportViewer	543
Grupa Components okna narzędziowego	544
Komponent BackgroundWorker	544
Komponent DirectoryEntry	544
Komponent DirectorySearcher	544
Komponent ErrorProvider	544
Komponent EventLog	545
Komponent FileSystemWatcher	545
Komponent HelpProvider	545

Komponent ImageList	546
Komponent MessageQueue	546
Komponent PerformanceCounter	546
Komponent Process	546
Komponent SerialPort	547
Komponent ServiceController	547
Komponent Timer	547
Komponenty i kontrolki do obsługi wydruku	547
Komponent PageSetupDialog	547
Komponent PrintDialog	547
Komponent PrintDocument	548
Komponent PrintPreviewControl	548
Komponent PrintPreviewDialog	548
Komponenty okien dialogowych	548
Komponent ColorDialog	548
Komponent FolderBrowserDialog	548
Komponent FontDialog	549
Komponent SaveFileDialog	549
Podsumowanie	549

Rozdział 36. Zaawansowane programowanie

interfejsu użytkownika551

Wprowadzenie do GDI+	551
Uzyskanie obiektu Graphics	552
Tworzenie przykładu „Witaj GDI+”	552
Rysowanie i wypełnianie kształtów	554
Wykorzystanie pędzli gradientowych	556
Tworzenie formularzy i kontrolki o niestandardowych kształtach	558
Wykorzystanie dziedziczenia wizualnego	559
Internacjonalizacja interfejsu użytkownika	560
Podsumowanie	563

Rozdział 37. Dostęp do danych w Windows Forms 2.0 565

Wiązanie danych za pomocą obiektów klasy DataSet z kontrolą typów	565
Wykorzystanie okienka Data Sources	566
Dodanie zbioru danych do formularza	566
Przykład dostępu do zbioru danych z kontrolą typów	568
Wprowadzenie do BindingSource	569
Wykorzystanie kontrolki BindingNavigator	572
Podstawy kontrolki BindingNavigator	572
Czynności wiązania danych wywoływane przez użytkownika	572
Praca z komponentem DataGridView	573
Podstawy DataGridView	574
Wykorzystanie kolumn ComboBox w kontrolce DataGridView	577
Zaawansowane dostosowanie komórek	578
„Niezwiązana” kontrolka DataGridView	581
Dostęp do danych obiektowych	581
Podstawy dostępu do danych za pośrednictwem obiektów	582
Wykorzystanie interfejsów IEditableObject i INotifyPropertyChanged	583
Wiązanie danych w relacji rodzic-potomek	586
Podsumowanie	586

Rozdział 38. Programowanie inteligentnych aplikacji klienckich 589

Praktyczne wykorzystanie Usług Web	590
Uzyskiwanie adresu URL usługi Web z UDDI	591
Zastosowanie nowego systemu ustawień aplikacji	591
Wsparcie dla pracy w trybie offline i online	593
Autoryzacja i uwierzytelnianie użytkowników	596
Wielowątkowe korzystanie z usług Web	598
Zastosowanie kontrolki BackgroundWorker	600
Podsumowanie	602

Rozdział 39. Wdrażanie aplikacji z użyciem technologii ClickOnce 603

Wprowadzenie do technologii ClickOnce	603
Publikowanie aplikacji ClickOnce	605
Rozpowszechnianie poprzez sieć Web lub udziały sieciowe	606
Umieszczanie na CD	606
Aplikacja uruchamiana bezpośrednio z sieci Web lub z udziału sieciowego	607
Instalacja aplikacji ClickOnce	607
Uaktualnianie aplikacji ClickOnce	610
Programowanie w przestrzeni nazw System.Deployment.Application	613
Podsumowanie	617

Rozdział 40. Zastosowanie usług korporacyjnych 619

Rejestrowanie komponentów usługowych	619
Rejestracja ręczna	620
Rejestracja automatyczna	620
Aktywacja w momencie wykonania i pule obiektów	622
Komponenty kolejkowane	624
Bezpieczeństwo oparte na rolach	625
Transakcje	627
Właściwości współdzielone	628
Luźno powiązane zdarzenia	630
Podsumowanie	633

Część VII Programowanie aplikacji korporacyjnych i rozproszonych 635**Rozdział 41. Technologia Remoting 637**

Przegląd technologii Remoting	637
Przedstawienie klasy MarshalByRefObject	638
Obiekty pojedynczego wywołania a obiekty singleton	639
Obsługa kanałów komunikacyjnych w technologii Remoting	641
Zastosowanie kanału IPC	641
Zastosowanie kanału TCP	645
Dzierżawy okresu istnienia	647
Technologia Remoting i typy ogólne	649
Podsumowanie	652

Skorowidz 653

Rozdział 2.

Wyrażenia

i struktury sterujące

U podstaw każdego języka programowania leży zdolność do spełniania dwóch różnych typów zadań: powtarzalności i logiki warunkowej. Powtarzalność pociąga za sobą zastosowanie konstrukcji językowych zwanych *pętlami* (ang. *loops*) w celu wielokrotnego wykonania tego samego zadania. Z kolei logika warunkowa wymaga sposobu zapisywania kodu, który będzie wykonany tylko wtedy, gdy zostaną spełnione określone warunki. Rozdział ten wprowadza podstawy umożliwiające nam realizację obu tych typów zadań z użyciem języka C#.

Jeśli posiadamy pewną znajomość języków programowania, pojęcia omawiane w tym rozdziale powinny okazać się znajome i z łatwością podchwycimy składnię C# w tym zakresie. Jeśli korzystamy z C# już od jakiegoś czasu, możemy jedynie przejrzeć pokrótce ten rozdział w ramach przypomnienia, zanim przejdziemy do dalszej części książki.

Rozgałęzianie i logika warunkowa

Każdy nowoczesny język programowania ma wsparcie dla rozgałęziania i logiki warunkowej. Umożliwia to naszej aplikacji wykonywanie różnych rzeczy w oparciu o bieżący strumień wejściowy, zdarzenia, warunki występowania błędów lub dowolne inne warunki, jakie można logicznie wyrazić. W tym podrozdziale najpierw ujrzymy, jak C# umożliwia formułowanie wyrażeń logicznych, oraz jak tworzyć bloki kodu, które są wykonywane warunkowo w oparciu o te wyrażenia, a następnie zobaczymy kilka skrótów i dodatków, jakie zawiera C#, jeszcze bardziej ułatwiających stosowanie pewnych popularnych typów instrukcji warunkowych.

Wprowadzenie do wyrażeń logicznych

Wyrażenie logiczne to instrukcja kodu, która ostatecznie przyjmie wartość `true` lub `false` (prawda lub fałsz). Na najbardziej podstawowym poziomie wszystkie wyrażenia logiczne (czy też warunkowe), bez względu na to, jak długie i złożone, przyjmą wartość `true` lub `false`.

Najprostszym ze wszystkich wyrażeń logicznych jest *równość*. Wyrażenie to jest stosowane do sprawdzenia, czy jedna wartość jest równoważna innej, czy też nie. Może to być coś nawet tak prostego, jak:

```
2 == 4
```

Wyrażenie to przyjmie wartość `false` (fałsz), gdyż 2 jest różne od 4. Może to być również coś tak złożonego, jak:

```
MyObject.MyProperty == YourObject.YourProperty
```

To wyrażenie może przyjąć dowolną wartość i zostanie to określone w czasie wykonywania programu. Jeśli jesteśmy zaznajomieni z C, C++ lub nawet C#, to wiemy, że `==` (podwójny symbol równości) jest logicznym operatorem boolowskim, natomiast `=` (pojedynczy znak równości) jest operatorem przypisania używanym do ustawiania wartości. Często błędy występujące w czasie kompilacji i wykonywania mają związek ze stosowaniem tych operatorów w nieodpowiednich miejscach.

Tabela 2.1 przedstawia operatory logiczne dostępne w języku C# oraz krótki opis ich przeznaczenia.

Tabela 2.1. Operatory logiczne w C#

Operator	Opis
&&	Operator iloczynu logicznego (ang. <i>Boolean AND</i>). Wyrażenie, w skład którego wchodzi, przyjmie wartość <code>true</code> (prawdę), jeśli <i>oba</i> podwyrażenia (po prawej i lewej stronie operatora) będą prawdziwe (spełnione). W przeciwnym razie zawsze będzie to <code>false</code> (fałsz).
	Operator sumy logicznej (ang. <i>Boolean OR</i>). Wyrażenie, w skład którego wchodzi, przyjmie wartość <code>true</code> , jeśli wyrażenie po lewej lub prawej stronie jest prawdziwe oraz wówczas gdy oba wyrażenia są prawdziwe.
!	Operator negacji (ang. <i>NOT</i>). Neguje wartość logiczną dowolnego wyrażenia, przy którym został zastosowany. Zatem jeśli <code>!</code> został użyty przy prawdziwym wyrażeniu, stanie się ono fałszywe. Analogicznie, jeśli <code>!</code> został użyty przy fałszywym wyrażeniu, stanie się ono prawdziwe.
^	Operator alternatywy wykluczającej (ang. <i>exclusive OR</i> , w wielu językach zapisywany jako <i>XOR</i>). Funkcjonuje on w taki sam sposób jak operator standardowej sumy logicznej (<code> </code>), za wyjątkiem tego, że jeżeli <i>obie</i> strony wyrażenia są prawdziwe, operator ten zwróci fałsz.
~	Operator ten wykonuje dopełnienie bitowe na swoim operandzie (argumente operacji). Oznacza to, że operator ten zamieni na odwrotny każdy bit tego operandu. Na wartości jednobitowej działa identycznie jak operator <code>!</code> , jednak na liczby takie, jak całkowite, dziesiętne itd. ma silniejszy wpływ.
&	Operator iloczynu bitowego (ang. <i>AND</i>). Zamiast iloczynu wykonywanego na dwóch wyrażeniach logicznych, operator ten działa na dwóch wartościach numerycznych. Liczby te są szeregowane, a mnożenie wykonuje się na każdej parze bitów sobie odpowiadających. Wynikiem tego działania jest zupełnie nowa liczba. Logika bitowa często ma zastosowanie, gdy przechowujemy długie listy wartości boolowskich jako liczby.
	Operator sumy bitowej (ang. <i>OR</i>). Podobnie jak to było z iloczynem bitowym, wykonuje on operację sumowania na każdej parze odpowiadających sobie bitów obu argumentów danego wyrażenia. Wynik jest zwrócony jako wartość liczbową.

Zastosowanie podstawowych instrukcji warunkowych

W poprzedniej części tego podrozdziału przedstawione zostały narzędzia potrzebne do budowania wyrażeń logicznych. Pozwolą nam one stwierdzić, czy dane wyrażenie jest prawdziwe, czy też nie. Gdy zachodzi taka potrzeba, musimy jeszcze zastosować pewne instrukcje warunkowe w naszym kodzie, aby móc wykonać jakieś znaczące operacje w oparciu o wynik interesujących nas wyrażeń logicznych. Ta część bieżącego rozdziału przedstawi podstawowe instrukcje warunkowe będące nieodzownym elementem każdego sensownego kodu napisanego w C#.

Zastosowanie instrukcji if/else

Struktura instrukcji if jest następująca:

```
if ( wyrażenie )
    blok_kodu
else if ( wyrażenie_1 )
    blok_kodu
else if ( wyrażenie_2 )
    blok_kodu
else
    blok_kodu
```

Obie sekcje `else if`, a także `else`, są opcjonalne i są wymagane jedynie, gdy chcemy w naszym kodzie wykonać pewne alternatywne zadania, kiedy główne wyrażenie logiczne (w naszym przypadku wyrażenie warunkowe) przyjmie wartość `false`.

Ponieważ każdy blok kodu w powyższym przykładzie może również zawierać swoje własne instrukcje `if`, możemy zagnieżdżać nasz kod warunkowy niemal tak głęboko, jak tylko chcemy. Jednakże w dobrym stylu, jak i w zgodzie z zaleceniem etykiety programisty jest unikanie głębokiego zagnieżdżania, gdyż czyni to nasz kod niezwykle trudnym do odczytu, a tym bardziej analizy.

Poniższy przykład instrukcji `if` obrazuje zastosowanie jej w sposób prosty, zagnieżdżony oraz z użyciem instrukcji `else`:

```
if (strInput == "Witam")
    Console.WriteLine("Powiedziałeś 'Witam'");

if (strInput2 == "Żegnam")
    Console.WriteLine("Powiedziałeś 'Żegnam'");
else if (strInput2 == "Tymczasem")
    Console.WriteLine("Nie powiedziałeś 'Żegnam', lecz 'Tymczasem'.");
else
{
    if (strInput3 == "Hola")
        if (strInput4 == "Senor")
            Console.WriteLine("Witaj!");
}
```


Zastosowanie instrukcji switch

Jeśli chcemy zbadać pojedynczą zmienną pod kątem całej listy możliwych wartości, korzystając jedynie ze standardowych wyrażeń kluczowych `if/else`, posłużymy się kodem zbliżonym do następującego:

```
if (val == 1)
    ...
else if (val == 2)
    ...
else if (val == 3)
    ...
else if (val == 4)
    ...
else
    ...
```

Choć spełni to swoje zadanie, nie będzie to jednak blok kodu ani elegancki, ani łatwo czytelny. W takiej sytuacji należałoby zastosować instrukcję C# o nazwie `switch`, która umożliwia przeprowadzenie kilku testów logicznych dla pojedynczego wyrażenia, co zostało pokazane za pomocą następującego przykładu:

```
switch (val)
{
    case 1:
        ...
        break;
    case 2:
        ...
        break;
    case 3:
        ...
        break;
    default:
        ...
}
```

Zastosowanie instrukcji goto

Instrukcja `goto` posiada dwa główne zastosowania. Pierwszym z nich jest przekazywanie kontroli z jednego bloku `case` do innego, w ramach instrukcji `switch`, natomiast drugim zastosowaniem jest przerywanie wykonywania pętli. O pętlach dowiemy się więcej w dalszej części tego rozdziału.

Poniżej znajduje się przykład instrukcji `switch`, w którym wykorzystano wyrażenie kluczowe `goto` do przekazania kontroli:

```
switch (val)
{
    case 1:
        ...
        break;
```

```
case 2:
    ...
    goto case 1;
    break;
case 3:
    ...
    goto case 1;
    break;
default:
    ...
}
```

Zastosowanie zaawansowanych instrukcji warunkowych

Jednym z najczęstszych zastosowań instrukcji `if/else` jest warunkowe wypisanie jakichś informacji lub wyświetlenie czegoś w oknie Windows albo na formularzu Web Forms. Przykładowo założmy, że chcemy wyświetlić słowo „odpowiednia”, jeśli marża wynosi więcej niż 20, lub „nieodpowiednia”, jeśli wynosi co najwyżej 20. Można by było w tym celu użyć `if/else` w następujący sposób:

```
if (profitMargin > 20)
    Console.WriteLine("Marża jest odpowiednia!");
else
    Console.WriteLine("Marża jest nieodpowiednia!");
```

Jest to jak najbardziej poprawne, jednak C# zawiera trójkrotny operator zwany *operatorem warunkowym*, umożliwiający zawarcie istoty kombinacji `if/else` w pojedynczej linii. Możemy zapisać powyższy blok kodu w następującej postaci:

```
Console.WriteLine("Marża jest "+
    (profitMargin > 20 ? "odpowiednia!" : "nieodpowiednia!"));
```

Nasz trójkrotny operator warunkowy ma następujący format:

```
wyrażenie ? zwróć_gdy_prawdziwe : zwróć_gdy_fałszywe
```

Zapętlanie i powtarzalność

Zapętlanie jest, obok logiki warunkowej i rozgałęziania, jednym z najczęściej występujących zadań wykonywanych przez wszystkie aplikacje, bez względu na ich typ. Obecnie nie znajdziemy żadnej produktywnej aplikacji napisanej w .NET, która by się nie składała z choćby kilku pętli.

Pętla to powtórzenie wykonania danego bloku kodu pewną ilość razy. Ilość ta jest określona przez typ użytej pętli. Podrozdział ten zawiera podstawowe informacje dotyczące tworzenia i stosowania pętli w C#.

Zastosowanie pętli for

Każda pętla for składa się z trzech części:

1. Część inicjalizująca. Jest to blok kodu wykonywany raz, na początku pętli for. Zmienne zadeklarowane w tej części są dostępne w obrębie pętli w czasie jej wykonywania, ale nie będą dostępne po wyjściu z niej. Część ta jest zwykle używana do inicjalizacji zmiennej lub zmiennych zliczających.
2. Warunek zakończenia. Część ta definiuje wyrażenie logiczne używane do określenia tego, jak długo pętla for ma być wykonywana. Pętla ta będzie się wykonywała tak długo, jak długo owe wyrażenie będzie prawdziwe. Będzie ono sprawdzane na początku każdego cyklu pętli.
3. Kod iteracyjny. Jest to część opcjonalna i może zawierać dowolny kod, jaki chcemy wykonać na końcu każdego powtórzenia pętli. Zwykle jest to miejsce, gdzie zwiększana jest zmienna zliczająca pętle (licznik).

Format pętli for jest następujący:

```
for ( część inicjalizująca; warunek zakończenia; kod iteracyjny )  
{  
    ...  
}
```

Zatem aby wywołać pętlę for, która wykona blok kodu pięć razy, można napisać:

```
for ( int x = 0; x < 5; x ++ )  
    Console.WriteLine("Wartość licznika to " + x.ToString());
```

Spowoduje to wyświetlenie następującego wyniku:

```
Wartość licznika to 0  
Wartość licznika to 1  
Wartość licznika to 2  
Wartość licznika to 3  
Wartość licznika to 4
```

Zastosowanie pętli while

Podczas gdy pętla for zwykle używana jest po to, by zapewnić indeksowanie cykli (zmienna zliczająca), pętla while wykonuje w kółko te same operacje, dopóki podane wyrażenie logiczne będzie prawdziwe. Format tej pętli jest następujący:

```
while ( wyrażenie ) { ... }
```

Jeśli chcemy wykonywać pewne czynności, dopóki pewna flaga (znacznik stanu) nie wskaże, że nie możemy już więcej ich wykonywać, wówczas nasza pętla while mogłaby wyglądać podobnie do:

```
bool canContinue = true;  
while (canContinue)  
{
```

```
... wykonuj czynności ...  
canContinue = ...  
}
```

Wyrażenie jest badane na początku każdego cyklu pętli i jeśli przyjmuje wartość `false`, wówczas kod w obrębie pętli nie zostanie wykonany. Często spotykanym błędem podczas używania pętli `while` jest zaniedbanie ustawienia warunku wyjścia z pętli. W naszym przykładzie jest nim zmienna `canContinue`. Jeśli nie ustawimy warunku wyjścia, pętla `while` będzie wykonywana w nieskończoność (lub dopóki nie wyłączymy naszej aplikacji „na siłę”).

Zastosowanie pętli do

Pętla `do` jest bardzo podobna do pętli `while`, poza faktem, że wyrażenie warunkowe jest sprawdzane na końcu każdego cyklu pętli zamiast na jego początku. Format pętli `do` jest następujący:

```
do  
{  
    ...  
} while ( wyrażenie );
```

Użycie pętli `do` zawsze gwarantuje, że kod w obrębie pętli zostanie wykonany przynajmniej raz. Wszystkie inne typy pętli mają sposobność niewykonania się ani razu wówczas, gdy warunek wejściowy nie zostanie spełniony. Oto przykład pętli `do`:

```
int x = 1;  
do  
{  
    Console.WriteLine(x);  
    x++;  
} while ( x < 10 );
```

Podsumowanie

Podczas pracy nad bazami danych, grafiką 3D, aplikacjami konsolowymi, aplikacjami opartymi na oknach Windows czy aplikacjami sieciowymi opartymi na Web Forms zawsze występuje kilka typów zadań, które muszą zostać wykonane, a należą do nich: zapętlenie oraz rozgałęzianie.

Zapętlenie i rozgałęzianie są głównymi mechanizmami zdefiniowanymi w każdym języku programowania. Bieżący rozdział pokazał nam, jak użyć języka `C#` do tworzenia złożonych rozgałęzień z użyciem wyrażeń warunkowych, oraz wiele sposobów na wykonanie zadań programowania iteracyjnego z użyciem pętli.

Rozdział ten zawiera również kilka informacji, które same w sobie mogą się wydawać mało użyteczne. Gdyby się jednak przyjrzeć możliwie każdemu problemowi programistycznemu, z jakim się zetkniemy w przyszłości, będziemy mieli twarde orzechy do zgryzienia, poszukując sposobu rozwiązania go i nie biorąc pod uwagę zastosowania pętli czy też rozgałęzień.