

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

Delphi 8 .NET. Kompendium programisty

Autor: Adam Boduch
ISBN: 83-7361-485-0
Format: B5, stron: 840



Poznaj najnowszą wersję Delphi i stwórz aplikację dla platformy .NET

W dobie tak dynamicznego rozwoju technologii informatycznych możliwość szybkiego stworzenia aplikacji jest dużym atutem. Dlatego powstają coraz bardziej rozbudowane środowiska programistyczne, umożliwiające skonstruowanie aplikacji z „cegiełek”, które można wykorzystywać wielokrotnie. Wśród tych środowisk programistycznych zasłużoną popularność zyskało Delphi, dostępne obecnie w wersji 8. Wygodny i czytelny interfejs, język programowania oparty na znanym i popularnym Pascalu, możliwość łatwego połączenia się z dowolną bazą danych oraz rozbudowany mechanizm tworzenia aplikacji wieloplatformowych to główne zalety tego środowiska.

Książka „Delphi 8 .NET. Kompendium programisty” to przewodnik po najnowszej wersji środowiska Delphi, uwzględniający jego nowe możliwości związane z tworzeniem aplikacji dla platformy .NET. Zawiera opis środowiska programistycznego i języka ObjectPascal. Przedstawia sposoby pisania aplikacji połączonych z bazami danych i korzystających z technologii XML. Opisuje również podstawowe zasady wykorzystywania w tworzonych programach protokołów sieciowych, takich jak HTTP i SMTP. Książka jest doskonała zarówno dla początkujących programistów Delphi, jak i dla tych, którzy chcą poszerzyć swoją wiedzę o zasady tworzenia aplikacji dla platformy .NET.

- Zasady tworzenia aplikacji dla platformy .NET
- Wizualne projektowanie aplikacji
- Mechanizm komponentów
- Projektowanie interfejsu użytkownika i menu
- Programowanie w języku ObjectPascal
- Korzystanie z procedur i funkcji
- Zasady projektowania obiektowego
- Programowanie oparte na zdarzeniach
- Wykrywanie i usuwanie błędów w aplikacjach
- Korzystanie z baz danych
- Język SQL
- Wykorzystywanie technologii XML w aplikacjach
- Programowanie sieciowe i ASP.NET
- Usługi sieciowe

Cennym źródłem informacji jest dodatek, zawierający praktyczne wskazówki dotyczące zasad pisania czytelnego i przejrzystego kodu.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl



Spis treści

Tematyka książki.....	19
Część I	21
Rozdział 1. Wstęp do programowania.....	23
Dawno, dawno temu.....	23
Czym jest programowanie?.....	24
Języki programowania.....	25
Asembler.....	26
Fortran.....	27
C.....	27
C++.....	27
Perl.....	27
PHP.....	28
Turbo Pascal.....	28
Java.....	28
C#.....	29
Kilka słów o kompilatorach.....	29
Działanie kompilatorów.....	30
Który kompilator wybrać?.....	31
Dzień programisty.....	31
Delphi.....	31
Object Pascal.....	32
Delphi — czy warto?.....	32
Czego wymagamy od Czytelnika?.....	33
Delphi 8 .NET.....	33
.NET.....	34
.NET Framework.....	34
Produkty dla programistów.....	35
Integracja językowa.....	35
Język pośredni IL.....	37
Wspólna specyfikacja języka.....	38

Wspólny system typów	38
Wspólny język wykonywania	38
Metadane	39
Działanie CLR	39
Zarządzanie aplikacją	40
Podzespoły	41
Biblioteka klas	41
Usługi sieciowe	41
Konkluzja	43
Instalacja Delphi	43
Wersje Delphi	44
Cennik	44
Wymagania Delphi	45
Instalacja	45
Polskie znaki	45
Jak się uczyć?	46
FAQ	47
Przydatne odnośniki	49
Podsumowanie	49
Rozdział 2. Projektowanie wizualne	51
Podstawowe okna Delphi	51
Welcome Page	52
Okno główne	53
Projektant formularzy	54
Paleta narzędzi	54
Inspektor obiektów	56
Menedżer projektu	57
Edytor kodu	59
Ukrywanie okna	60
Pierwszy projekt	61
Tworzenie projektu	61
Zapisywanie projektu	61
Uruchamianie programu	62
Zamykanie programu	63
Otwieranie projektu	63
Kompilacja projektu	63
Pliki projektu	64
Podstawowe paski narzędzi	65
Pasek Standard	65
Pasek View	66
Pasek Debug	66
Pasek Desktop	67
Pasek Custom	68
Pozostałe paski narzędzi	68
Repozytorium obiektów	68
Dodawanie projektu do repozytorium	69
Ustawienia repozytorium	70
Praca z paletą narzędzi	71
Umieszczanie komponentów na formularzu	71
Umieszczanie kilku komponentów naraz	72
Przycisk Categories	72
Szukanie obiektu	72

Przemieszczanie ikon	73
Menu palety narzędzi	73
Praca z komponentami	75
.NET	76
Windows Forms i VCL.NET	76
Komponent	78
Sterowanie komponentem	79
Praca z inspektorem obiektów	79
Edycja właściwości	80
Zdarzenia	82
Menu inspektora obiektów	82
Opcje inspektora obiektów	83
Projektant formularzy	83
Siatka pomocnicza	84
Usuwanie, kopiowanie i wklejanie	85
Zaznaczanie wielu komponentów	85
Menu projektanta formularzy	87
Pasek narzędziowy Align	87
Pasek narzędziowy Spacing	88
Pasek narzędziowy Position	89
Opcje projektanta formularzy	89
Projekty	90
Opcje projektu	90
Edytor WYSIWYG	97
Projektant strony WWW	98
Inspektor obiektów	98
Dodatkowe paski narzędzi	98
Edycja pozostałych plików	100
Opcje HTML w Delphi	100
Projektowanie interfejsu	101
Paski narzędziowe	101
Ikony dla paska narzędzi	103
Menu narzędzi	104
Menu podręczne	106
Pozostałe elementy interfejsu	106
Kilka wersji językowych programu	110
Tworzenie angielskiej wersji językowej	110
Tłumaczenie projektu	112
Kompilacja projektu	113
Opcje Translation Manager	114
Opcje środowiska	115
FAQ	116
Podsumowanie	117
Rozdział 3. Język Delphi	119
Aplikacje konsolowe	120
Zapisywanie projektu	120
Po kompilacji	121
Najprostszy program	122
Podstawowa składnia	122
Czytanie kodu	122
Wielkość liter	123
Pamiętaj o średniku!	123

Bloki begin i end.....	123
Dyrektywa program	124
Komentarze	125
Umiejętne komentowanie	126
Wypisywanie tekstu	126
Położenie instrukcji.....	128
Instrukcja Writeln	128
Zmienne.....	129
Deklaracja zmiennych.....	129
Typy zmiennych.....	130
Deklaracja kilku zmiennych	131
Przydział danych.....	132
Deklaracja zakresu danych	134
Restrykcje w nazewnictwie.....	135
Stałe.....	135
Domyślne typy stałych.....	135
Używanie zmiennych	136
Łączenie danych.....	137
Tablice danych	138
Tablice jako stałe	139
Tablice wielowymiarowe.....	140
Tablice dynamiczne	142
Polecenia Low i High.....	143
Operatory.....	144
Operatory przypisania	144
Operatory porównania	144
Operatory logiczne.....	145
Operatory arytmetyczne.....	146
Operatory bitowe	148
Pozostałe operatory	148
Instrukcje warunkowe	149
Instrukcja if..then.....	149
Instrukcja case..of	152
Instrukcja else	155
Programowanie proceduralne.....	158
Procedury i funkcje	158
Procedury	158
Funkcje.....	161
Zmienne lokalne.....	162
Parametry procedur i funkcji	163
Parametry domyślne.....	165
Przeciążanie funkcji i procedur.....	166
Przekazywanie parametrów do procedur i funkcji.....	167
Procedury zagnieżdżone	169
Własne typy danych	169
Tablice jako nowy typ.....	170
Aliasy typów	171
Rekordy	171
Przekazywanie rekordów jako parametrów procedury.....	172
Deklarowanie rekordu jako zmiennej	173
Instrukcja packed	173
Deklarowanie tablic rekordowych	174
Deklarowanie dynamicznych tablic rekordowych.....	174

Instrukcja wiążąca with	175
Programowanie strukturalne	176
Moduły	176
Tworzenie nowego modułu.....	176
Budowa modułu	177
Włączanie modułu	178
Funkcje wbudowane	178
Sekcje Initialization oraz Finalization	179
Dyrektywa forward	180
Konwersja typów.....	182
Rzutowanie	182
Parametry nieokreślone.....	184
Pętle	184
Pętla for..do	185
Pętla while..do.....	187
Pętla repeat..until	188
Procedura Continue.....	189
Procedura Break.....	189
Zbiory	190
Przypisywanie elementów zbioru	191
Odczytywanie elementów ze zbioru	192
Dodawanie i odejmowanie elementów zbioru	193
Typy wariantowe.....	193
VarType, VarTypeAsText	194
VarToStr	195
VarIs*	195
Pliki dołączane	195
Etykiety	196
Dyrektywy ostrzegawcze	197
Wstęp do algorytmiki	198
Schematy blokowe.....	198
Przykład — obliczanie silni	200
Funkcje konsolowe w .NET	203
Przykład — odczyt zawartości pliku	204
Efektywny kod	206
Instrukcje warunkowe.....	206
Typ Boolean w tablicach	207
Zbiory.....	209
Łączenie znaków w łańcuchach	209
FAQ.....	210
Podsumowanie	211
Rozdział 4. Delphi bardziej zaawansowane	213
Na czym polega programowanie obiektowe?	214
Biblioteki wizualne	214
Podstawowy kod formularza	215
Formularz w VCL.NET.....	217
Sekcja uses	217
Klasa	218
Zmienna wskazująca na klasę.....	218
Plik *.nfm.....	218
Generowanie kodu	219
Nazwy komponentów	219

Programowanie zdarzeniowe	220
Generowanie zdarzeń	220
Lista zdarzeń w inspektorze obiektów	222
Jedno zdarzenie dla kilku obiektów	223
Edytor kodu	224
Ukrywanie kodu	225
Makra	225
Code Insight	226
Code Completion	227
Menu podręczne edytora kodu	227
Opcje edytora kodu	229
Skrawki kodu	234
To-Do List	235
Generowanie komentarza TODO	236
Menu podręczne okna	237
Szablony kodu	237
Klasy	238
Składnia klasy	239
Do czego służą klasy?	239
Hermetyzacja	240
Dziedziczenie	240
Polimorfizm	241
Instancja klasy	241
Konstruktor	241
Destruktor	242
Pola	242
Metody	243
Tworzenie konstruktorów i destruktorów	244
Destruktry w .NET	247
Poziomy dostępu do klasy	251
Dziedziczenie	253
Przeciążanie metod	254
Typy metod	254
Prze definiowanie metod	256
Typy zagnieżdżone	260
Parametr Self	261
Brak konstruktora	261
Brak instancji klasy	263
Class helpers	264
Klasy zaplombowane	265
Słowo kluczowe static	266
Właściwości	267
Parametr Sender procedury zdarzeniowej	270
Przechwytywanie informacji o naciśniętym klawiszu	270
Obsługa parametru Sender	272
Operatory is i as	273
Metody w rekordach	274
Interfejsy	275
Przeładowanie operatorów	277
Jakie operatory można przeładować?	277
Deklaracja operatorów	279
Binary i Unary	279

Wyjątki	280
Słowo kluczowe try..except	280
Słowo kluczowe try..finally	281
Słowo kluczowe raise	282
Klasa Exception	283
Selektywna obsługa wyjątków	284
Zdarzenie OnException	284
Menedżer projektu	287
Model View	288
Data Explorer	288
Programowanie w .NET	288
Wspólny model programowania	288
Klasa System.Object	288
Główne przestrzenie nazw	290
Komponenty .NET	291
Atrybuty podzespołu	294
Mechanizm Reflection	297
Identyfikatory	310
Aplikacje .NET	311
Deassembler .NET	311
Global Assembly Cache Tool	312
WinCV	314
Narzędzie konfiguracji .NET Framework	315
PEVerify — narzędzie weryfikacji	315
Eksplorator modułów	316
Opcje eksploratora	317
FAQ	318
Podsumowanie	319
Rozdział 5. Wykrywanie błędów w aplikacjach	321
Rodzaje błędów	322
Opcje kompilatora	323
Częste błędy programisty	325
Niezainicjowane zmienne obiektowe	325
Zwalnianie obiektów	326
Tablice	326
Wskaźniki a .NET	327
Rejestry	328
Stos	328
Stera	329
Do czego służą wskaźniki?	329
Debugger Delphi	329
Interfejs Debug	330
Opcje projektu	330
Punkty przerwań	332
Polecenie Run to Cursor	337
Podgląd zmiennych	337
Inspektor śledzenia	340
Evaluate/Modify	341
Okno Call Stack	342
Okno Local Variables	343
Okno Thread Status	344
Okno Event Log	345

	Okno modułów.....	346
	Okno deasemblacji.....	346
	Polecenie Go to Address.....	347
	Okno Message View.....	347
	Praca krokowa.....	348
	Ikony na gutterze.....	348
	Przekraczanie i wkraczanie.....	349
	Opcje debugera.....	350
	Strona Borland .NET Debugger.....	351
	Zakładka Language Exceptions.....	351
	Zakładka Event Log.....	352
	Menu związane z debugerem.....	352
	FAQ.....	354
	Podsumowanie.....	354
Rozdział 6.	Migracja do .NET.....	355
	Czy warto przechodzić do .NET?.....	356
	Ewolucja platform programistycznych.....	356
	WinFX.....	357
	Brakujące komponenty.....	357
	Zmiany we właściwościach.....	358
	Elementy języka.....	358
	Wszystko jest klasą!.....	358
	Przestrzenie nazw.....	361
	Kompilacja warunkowa.....	363
	Brakujące elementy.....	364
	Łańcuchy w Delphi.....	373
	Komunikaty.....	378
	Destruktory.....	379
	WinForms.....	380
	Brak pliku *.dfm/*.nfm.....	380
	VCL i WinForms.....	385
	Platform Invoke.....	386
	Wywołanie tradycyjne.....	387
	Użycie atrybutu DLLImport.....	388
	Parametry wyjściowe.....	388
	Dane wskaźnikowe.....	390
	Pobieranie danych z bufora.....	391
	Kod zarządzany i niezarządzany.....	392
	Używanie funkcji Win32.....	393
	Marshaling.....	394
	Wady PInvoke.....	406
	.NET a obiekty COM.....	406
	Terminologia COM.....	406
	Mechanizm COM Callable Wrappers.....	408
	Przykładowy podzespół.....	408
	Utworzenie biblioteki typu.....	411
	Użycie biblioteki typu.....	412
	Korzystanie z klasy COM.....	414
	Kontrolki COM w aplikacjach .NET.....	417
	Aplikacje sieciowe.....	418
	Bazy danych.....	419
	FAQ.....	419
	Podsumowanie.....	420

Część II	421
Rozdział 7. Bazodanowa architektura Delphi.....	423
Czym jest baza danych?	423
Działanie baz danych	424
Rozwiązania alternatywne.....	425
Baza danych kontra własny mechanizm	426
Rodzaje baz danych.....	427
Bazy proste.....	427
Relacyjne bazy danych	427
Bazy danych typu klient-serwer.....	428
Wielowarstwowa architektura baz danych	429
Borland Database Engine	429
Sterowniki bazy danych.....	430
Zbiory danych	430
Komponenty bazodanowe.....	431
Praca z komponentami	435
Otwieranie i zamykanie zbioru danych.....	435
Przemieszczanie się wśród rekordów	435
Modyfikacja zawartości	437
Pola rekordu bazy danych	439
Edytor pól.....	442
Pola obliczeniowe	443
Pola przeglądowe	445
BDE Administrator	445
Tworzenie aliasu	446
Praca z tabelami	447
Tworzenie tabel.....	447
Tworzenie kolumn w kodzie programu	447
Filtrowanie tabel.....	449
Wykorzystanie właściwości Filter	450
Zdarzenie OnFilterRecord	451
Wyszukiwanie rekordów	452
Metoda Locate	453
Metoda FindKey	454
Przykładowa aplikacja.....	454
Założenia.....	455
Tworzenie interfejsu	455
Kod źródłowy aplikacji.....	460
Ćwiczenia dodatkowe	471
Zakładki.....	471
Pozostałe komponenty bazodanowe.....	472
Komponent TDbGrid	472
Komponent TDbNavigator	473
Komponent TDbText	473
Komponent TDbEdit.....	474
Komponent TDbMemo	474
Komponent TDbRichEdit	474
Komponent TDbImage	475
Komponent TDbCheckBox	475
FAQ.....	475
Podsumowanie	477

Rozdział 8.	dbExpress	479
	Architektura klient-serwer.....	479
	Klient.....	480
	Serwer	480
	Klient-serwer oraz bazy lokalne	481
	Język SQL	481
	Baza MySQL	482
	InterBase	482
	Schemat tworzenia aplikacji klient-serwer	483
	Analiza	483
	Projekt	483
	Budowa	483
	Programowanie w SQL-u.....	484
	Klient MySQL	484
	Tworzenie bazy danych	486
	Tworzenie tabel.....	487
	Zmiana struktury tabeli	493
	Indeksy	495
	Dodawanie rekordów	498
	Wyświetlanie informacji	500
	Uaktualnianie zawartości	505
	Usuwanie danych z tabeli	506
	Eksport i import bazy.....	506
	Aplikacja zarządzająca MySQL.....	507
	Praca z programem MySQL Control Center	508
	Praca z tabelami	509
	InterBase.....	511
	Uruchomienie.....	512
	Program IBConsole.....	512
	MySQL a InterBase	516
	Tworzenie tabel InterBase	525
	dbExpress	526
	dbExpress kontra BDE.....	527
	Połączenie z bazą danych.....	527
	Komponent TSQLDataSet.....	533
	Komponent TSQLMonitor.....	537
	Pozostałe komponenty dbExpress.....	540
	Dystrybucja aplikacji dbExpress.....	540
	Okno Data Explorer	540
	Wprowadzanie zmian w tabeli.....	541
	Edytor połączeń	541
	FAQ.....	542
	Podsumowanie	543
Rozdział 9.	IBX	545
	Dlaczego IBX?	545
	Zalety IBX	546
	Komponenty IBX	546
	Komponent TIBDatabase.....	547
	TIBDataSet	550
	Komponenty reprezentujące dane.....	553
	TIBSQLMonitor	554
	Komponenty administracyjne	554

Łączenie ze zdalnym serwerem.....	554
Przykład wykorzystania IBX	555
Tworzenie bazy danych	556
Piszemy aplikację Helion DB!.....	562
FAQ.....	566
Podsumowanie	566
Rozdział 10. ADO.NET i BDP.NET.....	567
Czym jest ADO?	567
Terminologia	568
Warstwy dostępu	569
Architektura baz danych.....	570
ADO.NET	570
Architektura ADO.NET	571
Źródło danych	571
Dostarczyciel danych	571
Zbiory danych	572
ADO.NET w praktyce.....	572
Łączenie ze źródłem ODBC	573
Łączenie ze źródłem OleDb	574
Wysyłanie zapytań (MS Access)	575
Sterowniki zarządzane.....	578
BDP.NET	579
Łączenie z bazą InterBase.....	579
Komponent BdpCommand	582
Komponent BdpDataAdapter.....	583
Wyświetlanie zawartości tabeli	585
Praca z danymi.....	585
Dystrybucja aplikacji BDP.NET	588
MySQL w ADO.NET	588
Korzystanie ze sterownika MySQL	589
Komponenty wizualne	595
FAQ.....	597
Podsumowanie	597
Część III.....	599
Rozdział 11. XML	601
Niezależność XML-a.....	602
XHTML.....	602
Budowa dokumentu.....	602
Prolog.....	603
Znaczniki.....	604
Atrybuty	606
Podstawowa terminologia.....	606
Węzeł główny	607
Komentarze	607
Przestrzenie nazw.....	607
Składnia przestrzeni nazw.....	608
Przestrzenie nazw i atrybuty	608

DTD.....	609
Deklaracja elementu.....	610
Deklaracja atrybutu.....	610
DTD w osobnym pliku.....	612
Encje tekstowe.....	613
XSD.....	614
Nagłówek XSD.....	614
Elementy XSD.....	615
Typy danych.....	615
Typy proste.....	616
XML a bazy danych.....	619
XSL.....	619
DOM.....	620
SAX.....	621
Korzystanie z System.XML.....	621
Ładowanie pliku XML.....	621
Odczyt dowolnego elementu.....	622
Odczyt wartości atrybutów.....	624
Tworzenie pliku XML.....	627
Eksport danych do postaci XML.....	632
Modyfikacja plików.....	634
FAQ.....	637
Podsumowanie.....	638
Rozdział 12. Programowanie sieciowe.....	639
Odrobina teorii.....	640
IP.....	640
TCP.....	641
Porty.....	641
HTTP.....	641
HTTPS.....	642
FTP.....	642
SMTP.....	643
Narzędzia.....	643
Pakiet Indy.....	644
Protokół HTTP.....	644
Przestrzeń nazw System.Net.....	655
Żądanie i odpowiedź.....	655
DNS oraz adres IP.....	661
Korzystanie z gniazd.....	663
Przestrzeń System.Net.Sockets.....	663
Tworzenie aplikacji_serwera.....	664
Tworzenie aplikacji-klienta.....	668
FAQ.....	672
Podsumowanie.....	672
Rozdział 13. ASP.NET.....	673
Dynamiczne strony WWW.....	673
CGI.....	674
PHP.....	675
ASP.....	676
ASP.NET.....	676

ASP i ASP.NET	677
Zmiany w kodzie.....	677
Kompilacja kodu.....	677
Migracja do ASP.NET	677
Zgodność ze standardem XHTML.....	680
Narzędzia.....	681
Edytor.....	681
Serwer	682
Instalacja ASP.NET	683
Co trzeba umieć?.....	684
ASP.NET w Delphi	684
Elementy interfejsu	685
Pierwszy projekt.....	686
Opcje ASP.NET	687
Web Forms	687
Przestrzeń nazw System.Web.UI.....	687
Praca z ASP.NET	688
Kontrolki działające po stronie serwera.....	688
Zdarzenia komponentów.....	690
Kontrolki Web Forms	693
Code Behind.....	698
Kontrolki użytkownika	700
Tworzenie kontrolki w Delphi	701
Komponenty .NET w ASP.NET	706
Konfiguracja stron ASP.NET	714
Sesje	716
Wysyłanie e-maili	721
Monitorowanie stron ASP.NET	722
Pamięć podręczna	723
Bazy danych w ASP.NET	724
Łączenie się z bazą.....	724
Kontrolki bazodanowe	726
Technologie internetowe	729
Komponenty HTML Producer.....	729
WebBroker.....	729
Internet Express	730
WebSnap	730
IntraWeb	730
ASP.NET	730
FAQ.....	731
Podsumowanie	731
Rozdział 14. Usługi sieciowe	733
Czym są usługi sieciowe?	733
Działanie usług sieciowych.....	735
HTTP.....	735
XML.....	735
Infrastruktura usług sieciowych.....	735
Użycie usług sieciowych.....	738
Wyszukiwarka google.com.....	738
Interfejs aplikacji	740
Ładowanie usługi sieciowej.....	740
Korzystanie z usługi Web	742

Usługi sieciowe w Delphi	748
Tworzenie usługi sieciowej.....	749
Podgląd usługi sieciowej.....	751
Usługa Web na stronie ASP.NET.....	754
Plik źródłowy *.asmx.....	757
Bezpieczeństwo usług sieciowych	757
Bazy danych	759
Projektowanie usługi.....	759
Sprawdzanie usługi sieciowej.....	766
Usługa sieciowa w aplikacji ASP.NET	767
FAQ.....	769
Podsumowanie	770
Dodatki.....	771
Dodatek A Akronimy	773
Dodatek B Spis przestrzeni nazw .NET.....	775
Dodatek C Słowniczek.....	779
Dodatek D Zasady pisania kodu.....	793
Stosowanie wcięć	794
Instrukcje begin i end	794
„Styl wielbłądzi” w nazwach procedur	795
Stosuj wielkie litery.....	795
Parametry procedur	796
Instrukcja if	796
Instrukcja case	796
Obsługa wyjątków.....	797
Klasy.....	797
Komentarze	797
Pliki i nazwy formularzy.....	798
Notacja węgierska	798
Czy warto?.....	799
Skorowidz	801

4.

Delphi bardziej zaawansowane

Jak dotąd mówiliśmy o programowaniu strukturalnym, proceduralnym, ale nie wspomnieliśmy o obiektach. Idea programowania obiektowego jest teraz niezwykle istotna oraz bardzo popularna. Większość języków wysokiego poziomu, np. Delphi, C++, C#, Java, Python, PHP i inne, umożliwia tworzenie obiektów. Na przykład dobry skądinąd C nie umożliwiał programowania obiektowego. I to jest wielka wada tych języków w obecnych czasach.

Rozdział niniejszy poświęcony będzie przede wszystkim opisowi tworzenia klas w Delphi. Omówimy także nowości wprowadzone do Delphi za sprawą .NET, a także zwrócimy uwagę na wiele narzędzi i opcji Delphi związanych z edycją kodu źródłowego.

W tym rozdziale:

- ✧ nauczysz się korzystać z edytora kodu,
- ✧ poznasz opcje edytora kodu,
- ✧ dowiesz się, czym jest programowanie obiektowe i jak tworzyć klasy,
- ✧ poznasz nowe elementy języka programowania wprowadzone za sprawą .NET,
- ✧ zapoznasz się z podstawowymi ideami programowania na platformie .NET,
- ✧ dowiesz się, czym są wyjątki,
- ✧ zostaną Ci przedstawione podstawowe narzędzia dołączone do pakietu .NET Framework.

Na czym polega programowanie obiektowe?

Programy rozrastają się coraz bardziej i bardziej. Tak samo jak kiedyś nie wystarczała idea programowania proceduralnego, teraz nie wystarcza już programowanie strukturalne.

Koncepcja programowania obiektowego pojawiła się już w latach 60. za sprawą języka Simula 67 zaprojektowanego przez naukowców z Oslo w celu przeprowadzania symulacji zachowania się statków. Jednakże idea programowania obiektowego swoją popularyzację zawdzięcza językowi SmallTalk. Połowa lat 80. to czas, kiedy programowanie obiektowe stało się dominującą techniką — głównie za sprawą C++. Wtedy to też w wielu innych językach pojawiła się możliwość tworzenia obiektów.

Można powiedzieć, że *klasa* to „paczuszka”, pewien element programu, który wykonuje jakieś funkcje. Klasa zawiera metody (procedury i funkcje) współdziałające ze sobą w celu wykonania jakiegoś zadania. Programowanie obiektowe przyczyniło się do tego, że takie „podprogramy” jak klasy mogą być wykorzystywane w wielu innych projektach — ułatwia to jeszcze bardziej zarządzanie i konserwację kodu.

Załóżmy, że napisałeś klasę do obsługi poczty (wysyłanie i odbieranie). Klasa może mieć procedury *Connect* (połącz), *SendMail* (wyślij e-mail), *Disconnect* (rozłącz). Z kolei procedura *Connect* może wywoływać inną, np. *Error* (która też jest procedurą znajdującą się w klasie), wyświetlającą błąd w razie niepowodzenia i zapisującą odpowiedni tekst w *dzienniku programu* (czyli, inaczej mówiąc, w *logach* — plikach z rozszerzeniem *.log). Teraz taką klasę możesz wykorzystać w wielu swoich aplikacjach — wystarczy skopiować fragment kodu i już gotowa jest obsługa błędów, łączenie itp. Taką klasę możesz udostępnić innym użytkownikom lub swoim współpracownikom. Taki inny użytkownik nie musi wiedzieć, jak działa klasa — ważne jest dla niego, co ona robi (wysyła e-maile). Użytkownik musi jedynie wiedzieć, że istnieje metoda *Connect*, która połączy go z danym serwerem oraz musi mieć świadomość obecności kilku innych procedur. To wszystko — nie interesuje go obsługa błędów — nie musi nawet zdawać sobie sprawy z jej istnienia.

Można by oczywiście utworzyć nowy moduł, a w module umieścić także procedury *Connect*, *SendMail* oraz *Disconnect*, *Error* i resztę potrzebnego kodu. Jednak w takim przypadku metody i zmienne (zmienne także mogą być elementami danej klasy) nie oddziałują na siebie w takim stopniu. Przykładowo, użytkownik korzystający z takiego kodu będzie miał dostęp do tych zmiennych, do których nie powinien go mieć. Będzie mógł też wywołać swobodnie procedurę *Error* — a nie powinien, bo może to spowodować niepożądane skutki. Dzięki klasom możesz sprawić, iż taka procedura *Error* nie będzie dostępna **poza** klasą; jej elementy (zmienne) też będą nie do odczytania przez przyszłego użytkownika.

Biblioteki wizualne

W rzeczywistości biblioteki takie jak VCL.NET (ang. *Visual Component Library*) to szereg modułów i klas współpracujących ze sobą. Dzięki idei programowania obiektowego to wszystko działa bardzo sprawnie i jest nad wyraz czytelne. Głównie dzięki funkcji *dziedziczenia*, która

polega na rozszerzeniu funkcjonalności klas. Klasa potomna może dziedziczyć po klasie bazowej, przejmując od niej całą funkcjonalność, ale jednocześnie można rozszerzyć ją o nowe funkcje. Tak samo zbudowana jest biblioteka VCL — w oparciu o dziedziczenie szeregu klas.

Podstawowy kod formularza

Poprzednio zajmowaliśmy się jedynie *Windows Forms*, ale od strony wizualnej, czyli głównie projektantem formularzy. Nie zajmowaliśmy się szczegółami takimi jak kod źródłowy formularza. Po otwarciu nowego projektu *WinForms* na pierwszym planie pojawi się czysty formularz. Klawiszem *F12* przejdź do edytora kodu. Podstawowy kod modułu źródłowego przedstawiony został na listingu 4.1.

Listing 4.1. Podstawowy kod modułu *WinForms*

```
unit WinForm2;

interface

uses
  System.Drawing, System.Collections, System.ComponentModel,
  System.Windows.Forms, System.Data;

type
  TwinForm2 = class(System.Windows.Forms.Form)
  {$REGION 'Designer Managed Code'}
  strict private
    /// <summary>
    /// Required designer variable.
    /// </summary>
    Components: System.ComponentModel.Container;
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    procedure InitializeComponent;
  {$ENDREGION}
  strict protected
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    procedure Dispose(Disposing: Boolean); override;
  private
    { Private Declarations }
  public
    constructor Create;
  end;

  [assembly: RuntimeRequiredAttribute(typeof(TwinForm2))]

implementation
```

```
{ $REGION 'Windows Form Designer generated code' }
/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWinForm2.InitializeComponent;
begin
    Self.Components := System.ComponentModel.Container.Create;
    Self.Size := System.Drawing.Size.Create(300, 300);
    Self.Text := 'WinForm2';
end;
{ $ENDREGION }

procedure TWinForm2.Dispose(Disposing: Boolean);
begin
    if Disposing then
    begin
        if Components <> nil then
            Components.Dispose();
        end;
        inherited Dispose(Disposing);
    end;
end;

constructor TWinForm2.Create;
begin
    inherited Create;
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent;
    //
    // TODO: Add any constructor code after InitializeComponent call
    //
end;

end.
```

Należy już na samym początku zaznaczyć, że składnia modułu *WinForms* znacząco różni się od standardowego VCL.NET (listing 4.2).

Listing 4.2. *Kod formularza w VCL.NET*

```
unit Unit1;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs;
type
    TForm1 = class(TForm)
    private
        { Private declarations }
    public
        { Public declarations }
    end;
end;
```

```
var
  Form1: TForm1;

implementation

{$R *.nfm}

end.
```

Na pierwszy rzut oka widać wyraźną różnicę pomiędzy kodem formularza *WinForms* a VCL.NET. Przede wszystkim ten drugi jest prostszy i krótszy.

Na razie omówimy podstawowe części kodu formularza w VCL.NET, gdyż jest on prostszy i może ułatwić użytkownikowi zrozumienie znaczenia poszczególnych elementów.

Formularz w VCL.NET

Moduły i ich budowa zostały szczegółowo omówione w poprzednim rozdziale poświęconym elementom języka Delphi. Zapamiętaj! Każdemu formularzowi odpowiada jeden moduł z odpowiednim kodem, lecz to stwierdzenie nie działa w drugą stronę — moduł niekoniecznie musi być formularzem.

Podczas projektowania wizualnego Delphi samodzielnie edytuje kod źródłowy, dodając lub zmieniając odpowiednie elementy — dzięki temu projektowanie staje się szybsze, a my nie musimy martwić się o szczegóły związane z dodaniem jakiegoś potrzebnego elementu.

Kod źródłowy formularza (modułu) składa się także ze standardowych sekcji i słów kluczowych typu: `unit`, `interface`, `uses`, `implementation`.

Sekcja `uses`

Podczas tworzenia nowego projektu Delphi doda do sekcji `uses` standardowe moduły konieczne do prawidłowego działania aplikacji. W tych modułach znajdują się deklaracje odpowiednich klas, dzięki czemu projekt może zostać skompilowany. Przykładowo, doprowadź sekcję `uses` do takiego wyglądu:

```
uses
  Windows;
```

Po próbie kompilacji Delphi już na początku wyświetli błąd: `[Error] Unit1.pas(9): Undeclared identifier: 'TForm'`. Komunikat mówi o tym, że nazwa `TForm` nie jest rozpoznawana dla kompilatora. Stało się tak dlatego, że usunęliśmy z sekcji `uses` moduł `Forms`, który zawierał deklarację typu `TForm`.

Klasa

Poniższy fragment w kodzie źródłowym jest deklaracją klasy, o której tyle mówiliśmy:

```
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

W tym momencie zadeklarowany został nowy typ — TForm1, który jest klasą wywodzącą się z kolei z klasy bazowej TForm.



Wskazówka

W VCL.NET nazwy wszystkich klas zaczynają się od litery T.

Zmienna wskazująca na klasę

Poniżej deklaracji klasy, jeszcze w sekcji interface, zadeklarowana jest standardowo zmienna, która wskazuje na nasz typ — TForm1:

```
var
  Form1: TForm1;
```

Plik *.nfm

Komponenty umieszczane na formularzu mogą mieć różne właściwości. Te dane muszą być gdzieś przechowywane. W VCL.NET miejscem, gdzie zapisywane są te informacje, jest plik *.nfm (w poprzednich wersjach Delphi plik miał rozszerzenie *.dfm), którego zawartość została przedstawiona na listingu 4.3.

Listing 4.3. Zawartość pliku *.nfm

```
object Form1: TForm1
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Tahoma'
  Font.Style = []
  Height = 240
  Width = 347
```

```
Left = 13
Top = 13
PixelsPerInch = 96
TextHeight = 13
end
```

Praktycznie nigdy nie zachodzi potrzeba, aby ręcznie edytować ten plik. To zadanie należy już do Delphi, które dokonuje zapisu i odczytu danych właśnie z tego pliku. Pliki **.nfm* zawierają informacje na temat formularza, jego właściwości, komponentów oraz właściwości komponentów.

W pliku źródłowym formularza znajduje się dyrektywa każąca kompilatorowi dołączyć do projektu plik **.nfm*:

```
{$R *.nfm}
```

Jest to specjalna instrukcja — na pierwszy rzut oka wyglądająca jak komentarz, jednak jest informacją dla kompilatora o włączeniu do projektu wszystkich plików z rozszerzeniem **.nfm*.

Generowanie kodu

Powróć do projektanta formularzy. Odszukaj na palecie narzędzi komponent `DBGrid` i umieść go na formularzu. Z powrotem przejdź do edytora kodu — zauważ, że Delphi w sekcji `uses` dodało nowe moduły: `Borland.Vcl.Grids` oraz `Borland.Vcl.DBGrids`, które są wymagane do prawidłowego skompilowania aplikacji. Dodało również nową linijkę w klasie `Form1`:

```
DBGrid1: TDBGrid;
```

Dzięki temu oszczędzamy na czasie, nie musząc wpisywać tego ręcznie.

Powyższa linia w klasie `Form1` identyfikuje komponent *typu* `TDBGrid` o nazwie `DBGrid1`. Delphi automatycznie nadaje własne nazwy tworzonym komponentem, przypisując im kolejne numery: `DBGrid1`, `DBGrid2` itd. Jeżeli na formularzu umieścisz kilka komponentów typu `TButton`, to Delphi nazwie je odpowiednio: `Button1`, `Button2` itd.

Nazwy komponentów

Nazwa komponentu jest symboliczna. Z punktu widzenia kompilatora nie jest ważne, czy komponent nazywa się `Button1` czy `ToJestMojFajnyButtonikNumer1`. Ważne jest jedynie to, aby nazwa była unikalna, a więc nie mogą istnieć dwa komponenty o tej samej nazwie. W nazwie komponentu nie mogą być użyte polskie znaki; nie może także zaczynać się ona od cyfry. W nazewnictwie komponentu istnieją takie same reguły co w deklaracji zmiennej.

Nazwę komponentu można zmienić, korzystając z inspektora obiektu. Znajdź właściwość `Name`, której możesz nadać nową wartość będącą nazwą komponentu. Po zmianie nazwy Delphi uaktualni również wszelkie wpisy w kodzie źródłowym związane z tą nazwą.



Wskazówka

Nigdy nie pozostawiaj domyślnych nazw komponentów, jakie nadaje Delphi (tzn. `Button1`, `Button2` itd.). W późniejszym czasie łatwo jest się pogubić w takim nazewnictwie. Staraj się nazywać komponenty opisowo — to znaczy tak, aby opisywały rolę, jaką pełnią. Jeżeli, przykładowo, masz komponent, który wyświetla informacje o autorze programu, nazwij go `btnAbout`; w przypadku przycisku, który będzie zamykał program — niech będzie to nazwa `btnClose`.

Programowanie zdarzeniowe

Programy napisane w Delphi są programami *zdarzeniowymi*, które wykonują pewne czynności pod wpływem nakazu użytkownika. Co to oznacza? Kiedy program jest uruchamiany, następuje jego załadowanie do pamięci, rezerwowanie pamięci itp. Następnie program taki czeka na zadania. Projektując nasze aplikacje wizualne, programujemy pewne zdarzenia, na jakie ma reagować program — np. kliknięcie przyciskiem myszy w obszarze komponentu. Jeżeli aplikacja odbierze takie zdarzenie (użytkownik kliknie w obszarze komponentu), podejmowane jest pewne działanie. To samo tyczy się innych przypadków — poruszanie myszą, wprowadzanie tekstu z klawiatury itp.

Generowanie zdarzeń

W kodzie źródłowym modułu nie możesz ręcznie umieszczać instrukcji wykonujących pewne czynności (pomijamy tu sekcję `initialization` oraz `finalization`) — wszystko musi być zawarte w procedurach zdarzeniowych.

1. Utwórz nowy projekt VCL.NET.
2. Umieść na formularzu komponent `TButton`.
3. Kliknij dwukrotnie w przycisk (komponent `TButton`).

W tym momencie Delphi powinno wygenerować w kodzie źródłowym zdarzenie i przenieść Cię do edytora kodu, natomiast kursor ustawić w ciele procedury:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Właśnie we wnętrzu tej procedury należy umieścić kod, który będzie wykonywany po naciśnięciu w trakcie działania programu przycisku. W naszym przykładzie spowodujemy tym zmianę właściwości komponentu.

Przykładowy program

Jak dotąd zmieniałeś właściwości komponentu jedynie z poziomu inspektora obiektów. Powinieneś wiedzieć, że jest to możliwe także z poziomu kodu źródłowego za pomocą operatora odwołania (.).

Nasz przykładowy program zmieniał będzie pozycję przycisku na formularzu. Nowa pozycja będzie losowana i ustawiana za każdym razem, gdy użytkownik kliknie przycisk.

Każda kontrolka wizualna w VCL.NET posiada właściwość `Width` (szerokość), `Height` (wysokość), `Left` (pozioma pozycja na formularzu), `Top` (pionowa pozycja na formularzu). Tak więc nadanie nowej wartości dla właściwości z poziomu kodu wygląda następująco:

```
Button1.Width := 100; // szerokość komponentu
```

Do konkretnych właściwości odwołujemy się za pomocą operatora odwołania (kropka), a wartości nadajemy tak jak zwykłym zmiennym.

Rozwiązanie

Przy każdorazowym naciśnięciu przycisku program powinien losować nową wartość z zakresu i ustawiać komponent na tej wylosowanej wartości. Pytanie: jaki to będzie zakres? Szerokość i wysokość formularza!

Listing 4.4 przedstawia kod całego modułu programu.

Listing 4.4. Kod modułu

```
unit Unit1;

interface

uses
  Windows, Messages, Variants, SysUtils, Graphics, Forms,
  Dialogs, System.ComponentModel, Borland.Vcl.Controls,
  Borland.Vcl.StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.nfm}
```



```
procedure TForm1.Button1Click(Sender: TObject);
var
  FormWidth, FormHeight : Integer;
begin
  { przypisz wartości opisujące szerokość i wysokość formularza }
  FormWidth := Form1.Width - 100;
  FormHeight := Form1.Height - 50;

  Randomize; // losowanie

  { nadanie nowych wartości właściwościom Left i Top }
  Button1.Left := Random(FormWidth);
  Button1.Top := Random(FormHeight);
end;

end.
```

Szerokość i wysokość formularza odczytujemy z właściwości `Width` oraz `Height`, a następnie przypisujemy te wartości do zmiennych zadeklarowanych wcześniej: `FormWidth` oraz `FormHeight`:

```
FormWidth := Form1.Width - 100;
FormHeight := Form1.Height - 50;
```

Zwróć uwagę, że dodatkowo odjęliśmy od tych wartości odpowiednio liczby 100 i 50, tak aby zachować pewien margines podczas losowania liczb.

Następnie wylosowane wartości posłużą jako nowe współrzędne położenia komponentu `TButton`:

```
Button1.Left := Random(FormWidth);
Button1.Top := Random(FormHeight);
```

Wiesz już, dlaczego odjęliśmy 100 i 50 od wartości zmiennej `FormWidth` i `FormHeight`? Zakładając, że szerokość formularza to 500, program będzie losował jedynie z zakresu od 0 do 400, tak aby nie wylosował liczby 500 — gdyby tak się stało, to komponent nie zmieściłby się całkowicie w oknie formularza (jego krawędź wystawałaby poza obszar formularza). Wszystko dlatego, że właściwość `Left` określa pozycję lewej krawędzi komponentu względem formularza.

Zmiany kosmetyczne

1. W inspektorze obiektów zmień właściwość `Width` oraz `Height` formularza na 500.
2. Zmień właściwość `Caption` przycisku (`Button1`) na `Kliknij mnie!`.

Działanie programu przedstawione zostało na rysunku 4.1.

Lista zdarzeń w inspektorze obiektów

Wspominaliśmy wcześniej o zakładce *Events* inspektora obiektów. Po wykonaniu wcześniejszego ćwiczenia na liście *Events* przy pozycji `OnClick` będzie widniał napis: `Button1Click` (rysunek 4.2).



Rysunek 4.1. Program w trakcie działania



Rysunek 4.2. Inspektor obiektów z wygenerowanym zdarzeniem OnClick

Po zaznaczeniu zdarzenia `OnClick` zostanie ono podświetlone na szaro. Po przeniesieniu kursora nad tekst, powinien on zmienić kształt. W tym miejscu po podwójnym kliknięciu zostaniesz przeniesiony do deklaracji zdarzenia `OnClick`. Tak samo jest z generowaniem innych zdarzeń — np. `OnKeyDown`.

Jedno zdarzenie dla kilku obiektów

Czasami może zająć potrzeba „podpięcia” tego samego zdarzenia `OnClick` (lub jakiegokolwiek innego) w kilku przyciskach lub innych komponentach.

W takim wypadku po zaznaczeniu wybranej pozycji, np. `OnClick`, po prawej stronie pojawia się strzałka, po naciśnięciu której rozwija się lista z dostępnymi w programie zdarzeniami. Możesz dla swojego komponentu wybrać dowolne zdarzenie.

Generowanie pozostałych zdarzeń

Możesz korzystać ze wszystkich zdarzeń, jakie w obrębie danego komponentu oferuje Ci Delphi. Przykładowo, zdarzenie `OnMouseMove` umożliwia Ci oprogramowanie zdarzenia przesuwania kursora nad obiektem. Innymi słowy, możesz odpowiednio na taką sytuację zareagować.

Jak odbywa się generowanie zdarzeń z poziomu inspektora obiektów? Zaznacz zdarzenie, niech to będzie `OnMouseMove`, klikając je. Po prawej stronie pojawi się lista rozwijalna, która na razie jest pusta. Przesuń kursor nad białe pole — kursor powinien zmienić swój kształt. W tym momencie kliknij dwukrotnie — spowoduje to wygenerowanie zdarzenia `OnMouseMove`.

```
procedure TMainForm.btnMainMouseMove(Sender: TObject; Shift: TShiftState;  
  X, Y: Integer);  
begin  
  
end;
```



Wskazówka

W niniejszej książce dla określenia nazw komponentów będą posługiwał się prawidłowym określeniem z literą *T* na początku — np. `TButton` czy `TLabel`. Są to prawidłowe nazwy dla komponentów VCL, chociaż wiele osób nadaje komponentom nazwy pozbawione tej litery — `Button`, `Label`.

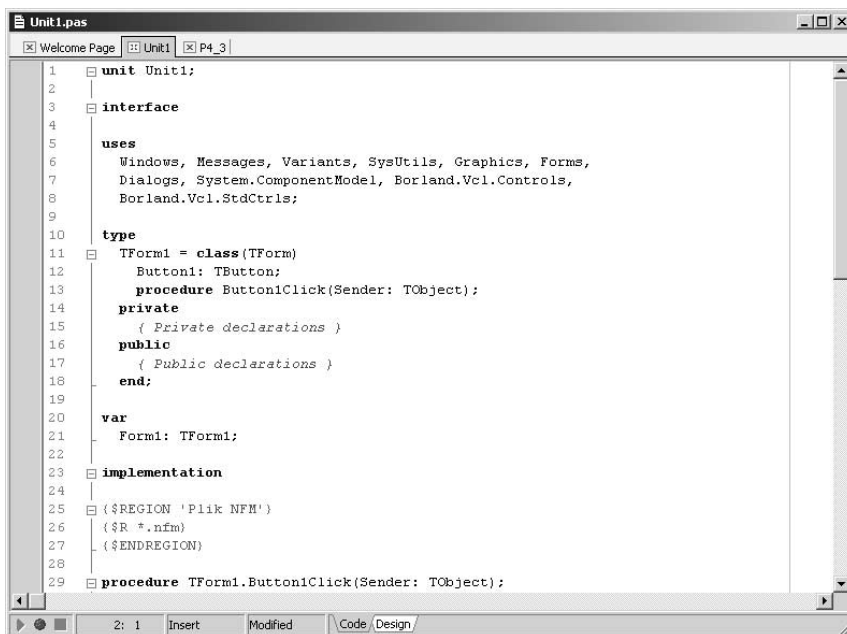
Jeżeli wygenerowane zdarzenie pozostanie puste, tj. nie będzie zawierało żadnego kodu, zostanie ono usunięte podczas kompilacji lub zapisu projektu.

Edytor kodu

Zanim przystąpimy do dalszej pracy, wypadałoby, abyś poznał główne narzędzie pracy używane w tym rozdziale — edytor kodu (rysunek 4.3).

Edytor kodu to zwykły edytor tekstów. Podzielony jest na zakładki, co daje możliwość jednoczesnej edycji wielu plików. W przypadku gdy mamy do czynienia z kodem źródłowym formularza, okno dodatkowo jest podzielone na dole na dwie zakładki — *Code* (pisanie kodu) oraz *Design* (projektowanie wizualne).

Trzeba przyznać, że w nowej wersji edytor kodu uległ zmianie — przede wszystkim dzięki dodaniu nowych opcji i możliwości.



Rysunek 4.3. Edytor kodu

Ukrywanie kodu

Jedną z nowości jest możliwość ukrywania fragmentu kodu. W takim przypadku po lewej stronie edytora wyświetlone zostaną małe ikonki, których kliknięcie spowoduje rozwinięcie ukrytego fragmentu. Jest to dość ciekawe rozwiązanie — umożliwi zwiększenie przejrzystości w kodzie i schowanie fragmentu, który nas nie interesuje.

Wszystko to odbywa się za sprawą dyrektywy `{ $REGION }`, którą należy wstawić przed blokiem, który chcemy ukryć. Wygląda to następująco:

```
{ $REGION 'Nazwa regionu' }
// kod, który chcemy ukryć
{ $ENDREGION }
```

Po wpisaniu takich instrukcji możliwe będzie ukrycie danych zapisanych pomiędzy tymi dyrektywami.

Makra

Mówiąc ogólnie, makra (również wprowadzone jako nowość w Delphi 8) umożliwiają „nagrywanie” pewnych operacji dokonanych w kodzie źródłowym, a następnie ich „odtworzenie”. Wszystkie trzy przyciski z tym związane znajdują się w lewym dolnym rogu edytora kodu.

Przeprowadź pewien eksperyment potwierdzający moje słowa: kliknij przycisk *Record Macro* (środkowy przycisk) i wpisz w edytorze kodu tekst:

```
ShowMessage('Nagrywamy makro...');
```

Po wpisaniu kliknij przycisk *Stop Recording Macro*. Od tej pory aktywny powinien być pierwszy przycisk — *Playback Macro*. Po jego kliknięciu w edytorze kodu ponownie zostanie wstawiona nagrana instrukcja.