

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

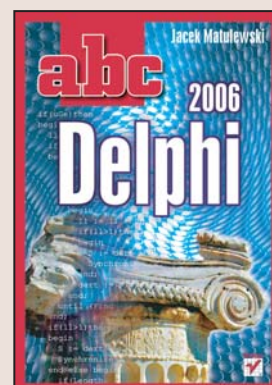
ABC Delphi 2006

Autor: Jacek Matulewski

ISBN: 83-246-0573-8

Format: B5, stron: 320

[Przykłady na ftp: 3950 kB](#)



Środowisko programistyczne Delphi od lat cieszy się zasłużoną popularnością wśród twórców oprogramowania. Potężne narzędzie programistyczne, oparte na popularnym języku Pascal, było prekursorem środowisk wizualnych, w których tworzenie aplikacji przypomina budowanie modelu z klocków. Kolejne wersje Delphi były wykorzystywane do tworzenia przeróżnych aplikacji – począwszy od prostych programików, a skończywszy na rozbudowanych systemach bazodanowych. Najnowsza wersja, oznaczona symbolem 2006, umożliwia tworzenie aplikacji dla platformy .NET oraz „tradycyjnych” aplikacji Win32.

Książka „ABC Delphi 2006” to wprowadzenie do programowania w tym środowisku. Na praktycznych przykładach przedstawia najnowszą wersję Delphi, język Object Pascal oraz filozofię tworzenia aplikacji na podstawie komponentów VCL. Czytając ją, poznasz środowisko programistyczne, elementy języka Object Pascal oraz zasady programowania strukturalnego i obiektowego. Nauczysz się budować własne aplikacje dla systemu Windows oraz wykorzystywać i tworzyć komponenty VCL. Zdobędziesz solidne podstawy do dalszej nauki programowania w Delphi.

- Struktura projektu w Delphi
- Typy danych i zmienne
- Instrukcje sterujące, pętle i wyrażenia warunkowe
- Programowanie obiektowe
- Wykrywanie i usuwanie błędów w kodzie
- Korzystanie z komponentów VCL
- Programowanie grafiki
- Operacje na plikach i drukowanie
- Zapisywanie informacji w rejestrze Windows
- Projektowanie komponentów

Poznaj jedno z najpopularniejszych narzędzi programistycznych





abc

SPIS TREŚCI

	Wstęp	11
I	Środowisko programistyczne Borland Developer Studio i język programowania Object Pascal	15
1	Poznajemy możliwości Delphi	17
	Platformy Win32 i .NET	18
	Co to jest Win32?	18
	Czym jest platforma .NET?	19
	Pierwszy projekt	20
	Projekt VCL Forms Application — Delphi for Win32	21
	Jak umieścić komponent na formie?	22
	Co to jest inspektor obiektów?	23
	Jak za pomocą inspektora obiektów zmieniać własności komponentów?	23
	Jak dopasować położenie komponentu?	26
	Jak umieszczać na formie wiele komponentów tego samego typu?	26
	Jak zaznaczyć wiele komponentów jednocześnie?	27
	Jak zaprogramować reakcję programu na kliknięcie panelu przez użytkownika?	27
	Domyślna metoda zdarzeniowa	28
	Polecenie ShowMessage	28
	CodeInsight, czyli nieoceniona pomoc w trakcie edycji kodu	29

Model zdarzeniowy projektowania aplikacji	30
Jak uruchomić projektowaną aplikację?	30
Jak przelączać między widokiem projektowania i edytorem?	31
Jak ustalić pozycję okna po uruchomieniu aplikacji?	31
Jak zmieniać własności obiektów programowo?	32
Jak zapisać projekt na dysku?	33
Oznaczenie zmian w kodzie	35
Pliki projektu Win32	35
Pierwsze podsumowanie	36
Ustawienia projektu	36
Jak zmienić tytuł i ikonę aplikacji?	36
Informacje o wersji aplikacji dołączane do skompilowanego pliku .exe	37
Déjà vu, czyli pierwszy projekt w wersji dla platformy .NET	40
Projekt VCL Forms Application — Delphi for .NET	40
Pliki projektu VCL.NET	43
Dystrybucja programów	44
Projekty VCL Forms Application — Delphi for Win32	44
Projekty dla .NET korzystające z biblioteki Windows Forms	44
Projekty VCL Forms Application — Delphi for .NET	44
Win32 albo .NET? — oto jest pytanie	45
Konfiguracja środowiska Delphi 2006	47
Okno postępu kompilacji	47
Automatyczne zapisywanie plików projektu	47
Modyfikowanie menu File/New	48
Edytor kodu	50
Opcje edytora	50
2 Analiza kodu pierwszej aplikacji, czyli wprowadzenie do języka Object Pascal	53
Wczytywanie istniejącego projektu	54
Jak wczytać wcześniej zapisany projekt do Delphi?	54
Plik Kolory.dpr	54
Moduł Unit1.pas	57
Jak wczytać moduł będący elementem projektu?	57
Czym jest moduł?	58
Sekcje modułu	59
Interfejs modułu Unit1	59
Implementacja	60
Pliki .dfm i .nfm	61
Kod źródłowy projektu dla platformy .NET	61
3 Typy zmiennych i instrukcje sterujące, czyli o tym, co każdy programista umieć musi	63
Podstawy	64
Równanie kwadratowe	64
Przygotowanie interfejsu	66
Deklarowanie zmiennych	67

Dygresja na temat typów rzeczywistych w Delphi	68
Konwersja łańcucha na liczbę	69
Obliczenia arytmetyczne i ich kolejność	71
Typ logiczny i operatory logiczne	72
Instrukcja warunkowa if	73
Jak wyłączyć podpowiadanie przez edytor szablonów instrukcji?	74
O błędach w kodzie i części else instrukcji warunkowej	75
Procedura Exit	78
Na tym nie koniec	79
Typy całkowite Delphi	79
Instrukcja wielokrotnego wyboru case	81
Procedura ShowMessage	83
Obsługa wyjątków	84
Czym są i do czego służą wyjątki?	84
Przechwytywanie wyjątków	85
Zgłaszanie wyjątków	87
Pętle	88
Pętla for	88
Pętla for w praktyce, czyli tajemnica pitagorejczyków	89
Uwaga o instrukcji goto, którą należy czytać z zamkniętymi oczami	92
Pętla repeat..until	92
Pętla while..do	93
Procedury Break i Continue	94
Podsumowanie	96
Typy złożone	96
Tablice	96
Pętla for..in..do	98
Tablice dwuwymiarowe	99
Definiowanie własnych typów	100
Tablice dynamiczne	100
Typy wyliczeniowe	101
Zbiory	102
Rekordy	106
Jak sprawdzić zawartość tablicy rekordów?	109
Instrukcja with	109
Kombinacja rekordów i typów wyliczeniowych	110
Kilka słów o konwersji typów	111
Łańcuchy	112
Dyrektywy preprocesora	114
Definiowanie bloków	114
Kompilacja warunkowa	116
Wskaźniki	117
Czym są wskaźniki?	117
Podstawowe konstrukcje	118
Wskaźniki w projekcie .NET	119
Do czego mogą służyć wskaźniki?	120
Wskaźniki — unikać, czy nie?	120

W domu:	121
Zdegenerowane równanie kwadratowe	121
Silnia	121
Imitacja pętli for	121
NWD	121
Ikony formy	122
Typ wyliczeniowy i zbiór	122
Rekordy	122
Instrukcja with	122
4 Programowanie strukturalne	123
Procedury, funkcje, moduły	124
Procedury	124
Definiowanie procedury	125
Interfejs modułu	127
Parametry procedur przekazywane przez wartość	129
Większa ilość parametrów	129
Wartości domyślne parametrów	130
Parametry przesyłane przez zmienną (referencje)	130
Funkcje	131
Funkcje imitujące globalne stałe	133
Zmienne proceduralne	133
W domu	134
Funkcje Silnia i NWP	134
Sekcje initialization i finalization modułu	134
5 Programowanie obiektowe	135
Pojęcia obiekt i klasa	135
Klasa	136
Referencje (zmienne obiektowe)	137
Tworzenie obiektów	138
Jeden obiekt może mieć wiele referencji	140
Interfejs i implementacja klasy	141
Definicja klasy	141
Projektowanie klasy — ustalanie zakresu dostępności pól i metod	142
Pola	144
Konstruktor klasy — inicjowanie stanu obiektu	144
Referencja Self	146
Tworzenie obiektów	146
Usuwanie obiektów z pamięci w projektach dla platformy Win32	147
Metoda Free w platformie .NET	147
Metoda prywatna	148
Zbiór metod publicznych udostępniających wyniki	149
Testowanie klasy	150
Przechwytywanie wyjątków	151
Zwiększona kontrola zakresu dostępności	151
Przestrzenie nazw	151
Nowe zakresy dostępności w klasach	152
Blokowanie dziedziczenia	153

W domu	153
Referencje	153
Referencja Self	154
Rozwój klasy TRownanieKwadratowe	154
Metody statyczne	154
Definiowanie operatorów w projektach .NET	155
6 Podstawy debugowania kodu	157
Debugger środowiska BDS	158
Ukryty błąd	158
Aktywowanie debugowania	159
Kontrolowane uruchamianie i śledzenie działania aplikacji	160
Zaawansowane techniki debugowania	161
Breakpoint	161
Obserwacja wartości zmiennych	162
Obsługa wyjątków przez środowisko BDS	163
Wyłączanie debugowania	165
7 Korzystanie z klas platformy .NET na przykładzie kolekcji	167
Kolekcje w Delphi	168
Kolekcja ArrayList	168
Kolekcja SortedList i inne	170
II Biblioteki komponentów VCL i VCL.NET ...	173
8 Podstawowe komponenty VCL/VCL.NET	175
Komponent TShape — powtórzenie wiadomości	176
Jak umieszczać komponenty na formie?	176
Jak modyfikować złożone własności komponentów za pomocą inspektora obiektów?	176
Jak reagować na zdarzenia?	177
Komponent TImage. Okna dialogowe	179
Automatyczne adaptowanie rozmiarów komponentów do rozmiaru formy	179
Jak wczytać obraz w trakcie projektowania aplikacji?	180
Konfigurowanie komponentu TOpenDialog	181
Jak wczytać obraz podczas działania programu za pomocą okna dialogowego?	181
Jak odczytać plik w formacie JPEG?	183
Kontrola programu za pomocą klawiatury	185
Wczytywanie dokumentu z pliku wskazanego jako parametr linii komend	186
Jak uruchomić projektowaną aplikację w środowisku BDS z parametrem linii komend?	187
Komponent TMediaPlayer	187
Odtwarzacz plików wideo	188
Panel jako ekran odtwarzacza wideo	190

	Wybór filmu za pomocą okna dialogowego w trakcie działania programu	190
	Odtwarzacz CDAudio	192
	Komponenty sterujące	192
	Suwak TScrollBar i pasek postępu TProgressBar	192
	Pole opcji TCheckBox	193
	Pole wyboru TRadioButton	194
	Niezależna grupa pól wyboru	196
	TTimer	197
	Czynności wykonywane cyklicznie	197
	Czynność wykonywana z opóźnieniem	198
	Aplikacja z wieloma formami	199
	Dodawanie form do projektu	199
	Dostęp do nowej formy z formy głównej	200
	Show versus ShowModal	202
	Zmiana własności Visible formy w trakcie projektowania ..	203
	Dostęp do komponentów z innej formy	204
	Właściciel i rodzic	205
	Własności Owner i Parent komponentów	205
	Zmiana rodzica w trakcie działania programu	207
	Co właściwie oznacza zamknięcie dodatkowej formy?	207
	Tworzenie kontrolki VCL w trakcie działania programu	208
	W domu	210
	Komponent TSaveDialog	210
	Komponenty TMemo, TRichEdit	210
	Komponent TRadioGroup	210
9	Więcej VCL	211
	Menu aplikacji	211
	Menu główne aplikacji i edytor menu	212
	Rozbudowywanie struktury menu	214
	Tworzenie nowych metod związanych z pozycjami menu	215
	Wiązanie istniejących metod z pozycjami menu	216
	Wstawianie pozycji. Separatory	216
	Usuwanie pozycji z menu	218
	Klawisze skrótów	218
	Ikony w menu	219
	Pasek stanu	220
	Sztuczki z oknami	222
	Jak spowodować, aby forma stopniowo zniknęła przy zamknięciu aplikacji	223
	Jak uzyskać dowolny kształt formy?	224
	Jak poradzić sobie z niepoprawnym skalowaniem formy w systemach z różną wielkością czcionki?	225
	Jak ograniczyć rozmiary formy?	226
	Jak przygotować wizytówkę programu (splash screen)?	227
	W domu	229
	Menu kontekstowe	229
	Pasek narzędzi	230

10	Prosta grafika	231
	Płótno i sztaluga	231
	Klasa TCanvas	231
	Odświeżanie formy. Zdarzenie OnPaint formy	232
	Linie	232
	Mieszanie kolorów	232
	Rysowanie linii	234
	ClientHeight i Height, czyli obszar użytkownika formy	236
	Okno dialogowe wyboru koloru TColorDialog	237
	Punkty	239
	Korzystanie z tablicy TCanvas.Pixels	239
	Negatyw	240
	Jak w projektach dla platformy Win32 umożliwić edycję obrazów z plików JPEG?	242
	Kilka słów o operacjach na bitach	244
	Własność TBitmap.ScanLine (platforma Win32)	246
	Wskaźniki IntPtr i klasa Marshal (platforma .NET)	247
	Inne możliwości płótna	248
	Tekst na płótnie	248
	Obraz na płótnie w projekcie Win32	251
	Obraz na płótnie w projekcie .NET	253
	W domu	254
11	Operacje na plikach i drukowanie z poziomu VCL i VCL.NET	255
	Automatyczne dopasowywanie rozmiaru komponentów	256
	Własność Align, czyli o tym jak przygotować interfejs aplikacji, który będzie automatycznie dostosowywał się do zmian rozmiaru formy?	256
	Komponent TSplitter	257
	Komponenty VCL pomagające w obsłudze plików	258
	Jak połączyć komponenty TDriveComboBox, TDirectoryListBox i TFileListBox żeby stworzyć prostą przeglądarkę plików?	259
	Jak filtrować zawartość komponentu TFileListBox?	259
	Prezentowanie nazwy katalogu wybranego za pomocą TDirectoryListBox na komponencie TLabel	259
	Prezentowanie na komponencie TLabel pliku wybranego w TFileListBox	261
	Jak z łańcucha wyodrębnić nazwę pliku, jego rozszerzenie lub katalog, w którym się znajduje?	261
	Wczytywanie plików graficznych wskazanych w FileListBox	263
	Przeglądanie katalogów w TFileListBox	264
	Obsługa plików z poziomu Object Pascala	266
	Tworzenie pliku tekstowego	266
	Dopisywanie do pliku	268
	Odczytywanie plików tekstowych	269
	Funkcja czy procedura	271

System plików	272
Operacje na plikach	273
Operacje na katalogach	275
Jak sprawdzić ilość wolnego miejsca na dysku?	275
Drukowanie „automatyczne”	276
Drukowanie tekstu znajdującego się w komponencie TRichEdit. Okno dialogowe TPrintDialog	277
Wybór drukarki z poziomu kodu aplikacji	279
Drukowanie „ręczne”	280
Tworzenie i przygotowanie modułu Drukowanie	280
Jak w trybie graficznym wydrukować tekst przechowywany w klasie TString?	281
Testowanie drukowania tekstu w trybie graficznym	285
Jak wydrukować obraz z pliku?	286
Dodawanie kodu źródłowego modułu do projektu	288
Powtórka z edycji menu aplikacji	289
Testowanie procedury drukującej obraz	289
W domu	290
Klasa TStringList	290
Rozwijanie procedur Drukuj	291
A Rozwiązania niektórych zadań	293
Rozdział 3	293
Zdegenerowane równanie kwadratowe	293
Silnia	295
Imitacja pętli for	296
NWD	296
Ikony formy	297
Typ wyliczeniowy i zbiór	298
Rekordy	298
Instrukcja with	299
Rozdział 4	299
Funkcja Silnia	299
Funkcja NWD	300
Sekcje initialization i finalization modułu	301
Rozdział 5	302
Referencje	302
Referencja Self	302
Rozwój klasy TRownanieKwadratowe	303
Metody statyczne	304
Rozdział 8	305
Komponent TSaveDialog	305
Komponent TMemo	306
Komponent TRadioGroup	307
Rozdział 9	308
Menu kontekstowe	308
Pasek narzędzi	308
Rozdział 10	309
Rozdział 11	310
Rozwijanie procedur Drukuj	310
Skorowidz	315

ANALIZA KODU PIERWSZEJ APLIKACJI, CZYLI WPROWADZENIE DO JĘZYKA OBJECT PASCAL

W poprzednim rozdziale przygotowaliśmy zgrabną aplikację, pozwalającą na zmienianie koloru panelu za pomocą trzech suwaków. Większość czynności wykonaliśmy z myszką w ręku, wpisując tylko dwie linie kodu. Jednak jeżeli zajrzemy do plików *Unit1.pas* i *Kolory.dpr* widocznych w katalogu projektu, to przekonamy się, że kodu powstało znacznie więcej. W tym rozdziale zanalizujemy ów kod, co da nam jednocześnie okazję do przedstawienia podstaw języka Object Pascal. Nie będzie to jednak dogłębne studium wszystkich elementów języka, które zobaczymy w tych plikach, bo wówczas pogubilibyśmy się w szczegółach.

Będzie to więc raczej turystyczny przewodnik po tej nowej galaktyce. Zatem i w tym rozdziale wiele razy będę obiecywać, że jakiś termin lub zagadnienie omówione będzie dokładnie w jednym z kolejnych rozdziałów.

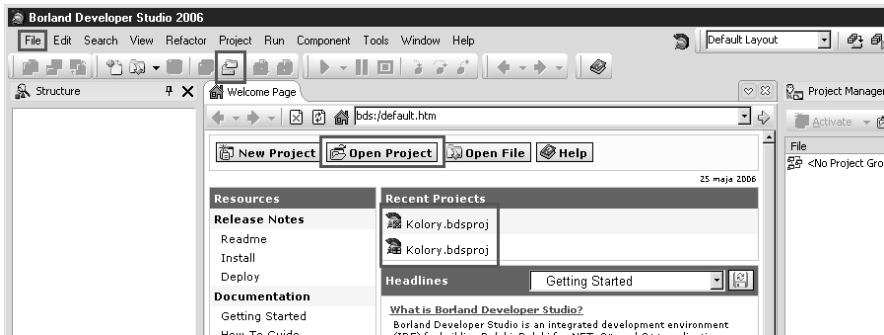
Wczytywanie istniejącego projektu

Jak wczytać wcześniej zapisany projekt do Delphi?

Otwórzmy projekt aplikacji, którą zajmowaliśmy się na początku poprzedniego rozdziału (projekt dla Win32). Jak to zrobić? Przede wszystkim należy zwrócić uwagę, czy uruchomiona przez nas wersja środowiska BDS zawiera moduł odpowiedzialny za przygotowywanie i kompilację projektu, który chcemy wczytać. Jeżeli uruchomiliśmy tylko moduł *Delphi for the Microsoft .NET Framework*, to nie uda nam się wczytać projektu typu *VCL Forms Application — Delphi for Win32*. Wówczas należy zamknąć BDS i ponownie je uruchomić, korzystając z ikony *Delphi for Microsoft Win32* lub *Borland Developer Studio 2006* w menu *Start*. Po uruchomieniu właściwej wersji środowiska mamy do wyboru co najmniej cztery sposoby na wczytanie zapisanego na dysku projektu. Po pierwsze w każdej chwili możemy na pasku narzędzi środowiska BDS (rysunek 2.1) kliknąć ikonę *Open Project* lub w menu *File* aplikacji wybrać pozycję *Open Project...* Jeszcze innym sposobem na wczytanie projektu jest użycie kombinacji klawiszy *Ctrl+F11*. W obu przypadkach pojawi się okno dialogowe pozwalające na wybór i wczytanie pliku projektu (pliku z rozszerzeniem *.bdsproj*). Ponadto, jeżeli poprzedni projekt został zamknięty (tzn. z menu *File* wybraliśmy polecenie *Close All*), to w głównej części środowiska widoczna jest pokazana na rysunku 2.1 lista kilku ostatnich projektów. Niestety, jeżeli ktoś, tak jak ja, jest zbyt leniwy, żeby opisowo nazywać projekty, to zobaczy jedynie listę kilku plików *Project1.bdsproj*. Wówczas pomóc może informacja o dokładnym położeniu pliku, która pojawi się, gdy chwilę potrzymamy kursor nad nazwą projektu. W dowolny z opisanych powyżej sposobów wczytajmy projekt aplikacji *Kolory.bdsproj*, któremu towarzyszy złota ikona symbolizująca aplikację dla Win32.

Plik *Kolory.dpr*

Po wczytaniu zobaczymy nie ten plik, z którym pracowaliśmy w poprzednim rozdziale (*Unit1.pas*), tylko widoczny na listingu 2.1 plik *Kolory.dpr*. Jest to główny plik programu, od niego rozpoczyna się wykonywanie kodu całego projektu.



Rysunek 2.1. Wszystkie drogi prowadzą do...

Listing 2.1. Główny plik programu *Kolory*

```

program Kolory;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.Title := 'Kolory';
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.

```

Skoro już mamy go przed oczami i jest taki ważny, to poświęćmy mu chwilę. Przede wszystkim zwróćmy uwagę, że plik ten zaczyna się od słowa kluczowego `program` i następującej po nim nazwie aplikacji *Kolory*. Tak zaczynają się wszystkie programy napisane w Pascalu. Tylko jeden plik projektu może rozpoczynać się w ten sposób, i w Delphi jest nim właśnie plik `.dpr`. W kolejnej linii widzimy słowo kluczowe `uses`, po którym wymienione są dwa moduły. Moduł *Forms* z biblioteki VCL oraz dobrze nam znany moduł *Unit1*. Taką nazwę nosi moduł zapisany w poprzednim rozdziale do pliku *Unit1.pas*. O ile położenie modułu *Forms* jest znane kompilatorowi Delphi, to plik, w którym znajduje się moduł *Unit1*, jest jawnie wskazywany. Służy do tego konstrukcja `Unit1 in 'Unit1.pas'` (moduł *Unit1* w pliku *Unit1.pas*). Czym jest **moduł**, możemy już chyba sobie wyobrazić. Mówiąc najprościej, jest to osobny plik z kodem źródłowym. Ale to tylko wstępna definicja.

To co widoczne jest w dalszej części, a więc {Form1}, jest tylko **komentarzem**. Każdy tekst znajdujący się pomiędzy nawiasami klamrowymi, z pewnymi wyjątkami, jest zwyczajnie ignorowany przez kompilator. Ignorowany jest więc napis „Form1”, ale już nie znajdujący się za nawiasem średnik ;. Nawiasy mogą znajdować się w różnych liniach, co oznacza, że w ten sposób można „zakomentować” również wiele linii jednocześnie.

Innym typem komentarza jest //. Wszystko, co znajduje się po podwójnym znaku slash aż do końca linii, jest również ignorowane przez kompilator. Wobec tego omawiana linia mogłaby z równym skutkiem mieć następującą postać:

```
Unit1 in 'Unit1.pas';    //Form1
```

W kolejnej linii znajduje się coś dziwnego ({\$R *.res}), co wygląda jak komentarz, ale nim nie jest. Jest to dyrektywa prekompilatora. Ta konkretna dyrektywa wczytuje do projektu plik zasobów .res, czyli plik, w którym zapisana jest np. ikona aplikacji oraz informacje o wersji i nazwie produktu, które określiliśmy w poprzednim rozdziale. Więcej o dyrektywach prekompilatora dowiemy się w następnym rozdziale. Tu jednak możemy uściślić wiedzę o wieloliniowych komentarzach. Nie jest komentarzem wyrażenie, w którym za lewym nawiasem klamrowym znajduje się znak dolara. To jest dyrektywa.

I w ten sposób docieramy do zasadniczej części, rozpoczynającej się od słowa kluczowego begin, a zakończonej słowem kluczowym end.. Między nimi znajdują się instrukcje wykonywane przez program. Z tego, co widzimy, wszystkie cztery wykonywane w tym miejscu polecenia są związane z obiektem Application. Jest to obiekt tworzony automatycznie (jego definicja znajduje się w module *Forms*, który jak pamiętamy został wymieniony w sekcji uses). Obiekt ten reprezentuje naszą aplikację. Po pierwsze wywoływana jest jego metoda *Initialize*, która w naszym projekcie tak naprawdę niczego ciekawego nie robi (właściwie nie robi nic)¹. Następnie zmieniana jest własność *Title* aplikacji na *Kolory*. Linia ta pojawiła się w momencie, w którym ustalaliśmy tytuł aplikacji za pomocą okna opcji projektu.

Znacznie ważniejsze są dwie kolejne linie. Polecenie *Application.CreateForm* (*TForm1*, *Form1*); tworzy bowiem obiekt formy i zapisuje odwołanie do niego do zmiennej o nazwie *Form1* zadeklarowanej w module *Unit1*. Utworzony obiekt formy jest instancją klasy *TForm1*. Brzmi to trochę zagadkowo, ale jest proste. Relacja między obiektem a klasą jest taka sama, jak między zmienną i jej typem.

¹ Zwróćmy uwagę, że wywołanie funkcji, procedur i metod, które nie przyjmują żadnych argumentów, nie wymaga w Object Pascalu dodawania pustych nawiasów. Choć jest to dopuszczalne.

Tyle że klasa może być projektowana przez nas, a typy zmiennych są wbudowane w kompilator Delphi. W szczególności klasa `TForm1`, opisująca formę aplikacji *Kolory*, jest zdefiniowana w module *Unit1*. Jej definicja obejmuje wszystkie zmiany, jakie wprowadziliśmy w widoku projektowania (dodaliśmy do niej panel i suwaki). Zawiera także zdefiniowane przez nas dwie metody zdarzeniowe (są jej metodami składowymi). Projektowaniu kodu klasy `TForm1` był w zasadzie poświęcony cały poprzedni rozdział.

I wreszcie ostatnia linia programu zawiera wywołanie metody `Application.Run`. Uruchamia ona główną pętlę programu. Była ona już kilkakrotnie wspomniana w pierwszym rozdziale. Pętla ta jest odpowiedzialna za odbieranie od systemu Windows komunikatów dotyczących aplikacji i generowania na ich podstawie zdarzeń, które powodują uruchamianie zdefiniowanych przez nas metod zdarzeniowych. Jej wykonywanie trwa, dopóki użytkownik aplikacji nie postanowi zamknąć formy i w ten sposób zakończyć działania aplikacji.

Moduł `Unit1.pas`

No dobrze. Plik *Kolory.dpr* został całkowicie i automatycznie utworzony przez Delphi. Czytelnika zapewne bardziej interesuje moduł *Unit1*, nad którym spędziliśmy kilka miłych chwil, wykonując czynności opisane w poprzednim rozdziale. Jak się do niego dostać?

Jak wczytać moduł będący elementem projektu?

Po wczytaniu zapisanego na dysku projektu do Delphi powstały dwie zakładki. Pierwsza, na której znajduje się plik *Kolory.dpr*, i druga, z plikiem, w którym zdefiniowana jest główna forma projektu, zazwyczaj *Unit1.pas*. Zatem jedyne, co musimy zrobić, to zmienić zakładkę w górnej części okna edytora na *Unit1*. Pojawi się wówczas widok projektowania formy zdefiniowanej w tym pliku. Jeżeli chcemy obejrzeć kod pliku *Unit1.pas*, musimy nacisnąć klawisz *F12*, który, co już mamy dobrze opanowane, służy do przełączania między widokiem projektowania a edytorem kodu.

Jeżeli jednak taka zakładka nie pojawiła się lub ją niechcący zamknęliśmy, możemy wesprzeć się oknem projektu (numer 2 na rysunku 1.1). Klikając dwukrotnie nazwę pliku *Unit1.pas*, wczytamy go do edytora.

Czym jest moduł?

Edytor kodu zawiera znajomy kod pliku *Unit1.pas* (listing 2.2). W jego dolnej części widoczne są napisane przez nas metody *Panel1Click* i *ScrollBar1Change*. Ale zacznijmy od początku. A na początku jest słowo kluczowe *unit*, a po nim nazwa *Unit1* — ta, którą widzieliśmy w sekcji *uses* w pliku *Kolory.dpr*. Słowo *unit* sygnalizuje, że plik, z którym mamy teraz do czynienia, jest modułem, a więc nie samodzielny programem, który zaczynałby się od słowa kluczowego *program*, a tylko oddzielną jednostką kodu, w której zdefiniowane mogą być procedury i funkcje oraz klasy i ich metody (formalnie nazywane procedurami i funkcjami składowymi). Nazwą tego modułu jest *Unit1*, wskazuje ją pierwsza linia kodu.

Listing 2.2. Pełen kod modułu, w którym zdefiniowana jest forma

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    ScrollBar1: TScrollBar;
    ScrollBar2: TScrollBar;
    ScrollBar3: TScrollBar;
    procedure ScrollBar1Change(Sender: TObject);
    procedure Panel1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Panel1Click(Sender: TObject);
begin
  ShowMessage('Witaj Świecie!');
end;
```

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
  panel1.Color:=RGB(scrollbar1.Position,scrollbar2.Position,scrollbar3.Position);
end;

end.
```

Dla osób rozpoczynających programowanie pojęcia procedury, funkcji i metody, są one oczywiście nowościami i w związku z tym częste ich używanie może łatwo spowodować lekki zawrót głowy. Bez obawy, wszystkie te terminy zostaną jeszcze wyjaśnione w rozdziale czwartym. Już mamy jednak pewną intuicję, czym one są — *de facto* dwie metody napisaliśmy przecież w poprzednim rozdziale.

Sekcje modułu

Każdy moduł podzielony jest na dwie części: interfejs i implementację. Są to nie tylko nazwy części modułów, ale i podstawowe terminy programowania strukturalnego. **Interfejs** to zbiór tych procedur, funkcji i klas, które są udostępniane przez ten moduł innym modułom, jego część publiczna. Wszystkie deklaracje, które znajdują się w sekcji *interface* modułu *Unit1*, są widoczne np. z pliku *Kolory.dpr*. W szczególności dostępna jest tam klasa *TForm1*, której definicja jest też w sekcji *interface*. Natomiast **implementacja** to część prywatna modułu. W niej znajdują się kody procedur, funkcji i metod. Mogą być tu również zadeklarowane pomocnicze zmienne. Jeżeli funkcja zdefiniowana jest w sekcji implementacji i nie jest zadeklarowana w sekcji interfejsu, nie będzie widoczna poza modułem.

Poza tym moduł może zawierać jeszcze sekcję *initialization*, która służy do wykonywania poleceń w momencie wczytywania modułu do pamięci przy starcie programu, oraz sekcję *finalization*, która pozwala na określenie czynności związanych z kończeniem pracy modułu przy zamykaniu programu. Może to być na przykład otwieranie pliku rejestrowania zdarzeń aplikacji i jego zamykanie. Sekcje te nie są domyślnie tworzone w modułach Delphi, i w tej książce, poza prostym ćwiczeniem w rozdziale 4, nie użyjemy ich ani razu.

Interfejs modułu Unit1

W interfejsie modułu *Unit1* widzimy trzy sekcje: *uses*, *type* i *var*. W pierwszej wymienione są inne moduły, których zawartość ma być widoczna w trakcie kompilacji bieżącego modułu.

Sekcja *type* zawiera natomiast definicje nowych typów, które są zdefiniowane w module *Unit1*. W naszym przypadku zdefiniowany jest tam typ *TForm1*, który, który rozpoczyna się od linii:

```
type
  TForm1 = class(TForm)
```


Po niej następują definicje zmiennych: `Panel1`, która związana jest z panelem, i trzy zmienne: `ScrollBar1`, `ScrollBar2` i `ScrollBar3`, reprezentujące suwaki. Widoczne są również deklaracje dwóch metod zdarzeniowych, które stworzyliśmy, a więc `Panel1Click` i `ScrollBar1Change`. One również są zadeklarowane w klasie formy — wobec tego też są w jakiś sposób z nią związane. To jest właśnie definicja klasy `TForm1`. Zmienne zdefiniowane w jej obrębie nazywane są **polami**. `Panel1` jest zatem polem klasy `TForm1`. Natomiast funkcje i procedury zdefiniowane w obrębie klasy będziemy nazywać **metodami**. Nie chciałbym jednak teraz dalej rozwijać tematu klas — na to przyjdzie czas w rozdziale 5.

Ostatnia sekcja interfejsu modułu *Unit1* to sekcja `var`. Zawiera ona definicję jednej tylko zmiennej. Jest nią zmienna o nazwie `Form1` typu `TForm1`. W rzeczywistości już tę zmienną poznaliśmy. Jeżeli przypomnimy sobie polecenie tworzące formę z pliku *Kolory.dpr*:

```
Application.CreateForm(TForm1, Form1);
```

to okaże się, że właśnie do zmiennej `Form1` zapisana została referencja do utworzonego obiektu formy. Zmienna ta reprezentuje więc obiekt formy, który widzimy w postaci okna po uruchomieniu aplikacji.

Implementacja

W sekcji implementacji mogą znajdować się te same podsekcje, co w interfejsie, a więc `uses`, `type` i `var`. Jednak w tym przypadku typy zdefiniowane w sekcji `type` i zmienne zadeklarowane w sekcji `var` będą lokalne, czyli niedostępne poza modulem. Będą mogły być wykorzystane jedynie w sekcji implementacji tego konkretnego modułu. Podobnie moduły wymienione w sekcji `uses`, która może tu być umieszczona, będą widoczne jedynie w kodzie z implementacji tego modułu. Możliwość tworzenia lokalnej sekcji `uses` okaże się bardzo ważna; pozwoli bowiem zapobiec zapętleniu modułów, co zdarza się, gdy np. w projekcie mamy więcej form (każda definiowana jest w osobnym module), które chcą być dla siebie nawzajem widoczne. Problem ten zostanie szerzej omówiony w rozdziale 8.

W interfejsie naszego modułu nie ma jednak żadnej z wymienionych wyżej sekcji, a jest tylko kolejna dyrektywa prekompilatora i definicje dwóch metod, które zdefiniowaliśmy w poprzednim rozdziale. Skoro już wiemy, że klasa `TForm1` jest właścicielem tych metod, to zwróćmy uwagę, że w odróżnieniu do ich deklaracji w klasie, które widzieliśmy w sekcji `interface`, tu ich nazwy poprzedzone są nazwą klasy, do której należą. Mamy więc np.

```
procedure TForm1.Panel1Click(Sender: TObject);
```

To właśnie odróżnia definicje procedur składowych, które umówiliśmy się nazywać po prostu metodami, od zwykłych procedur.

Pliki .dfm i .nfm

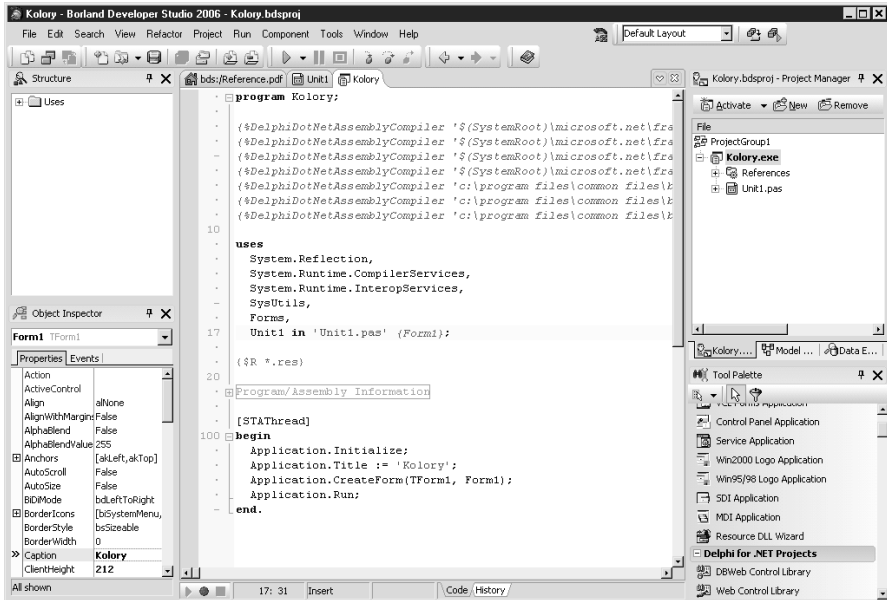
Kolejna dyrektywa preprocesora, która znajduje się w interfejsie modułu, jest bardzo podobna do tej, która dołączała do projektu plik zasobów. Tym razem jednak dołączany jest plik z rozszerzeniem *.dfm* (w projektach *.NET* — *.nfm*). Jest to plik zawierający wartości własności formy i umieszczonych na niej komponentów, jakie określiliśmy za pomocą inspektora obiektów i na podglądzie formy w widoku projektowania. Z łatwością możemy ten plik obejrzeć, korzystając choćby z Notatnika Windows, jest to bowiem w nowych wersjach Delphi plik tekstowy. Można go również obejrzeć w środowisku Delphi. W tym celu należy przejść do widoku projektowania (*F12*) i kliknąć prawym klawiszem na podglądzie formy, aby rozwinąć menu kontekstowe. Z tego menu wybieramy pozycję *View as Text*. Zamiast podglądu formy zobaczymy wówczas równoważny jej zbiór własności. To jest właśnie zawartość pliku *Unit1.dfm*. Zawiera ona to wszystko, co można skonfigurować w formie w widoku projektowania, a co bezpiecznie jest usunąć sprzed oczu programisty, a więc z edytowanego pliku modułu *Unit1.pas*. I tak naprawdę lepiej w pliku *.dfm/.nfm* nie grzebać ręcznie — do jego edycji służy widok projektowania. Aby powrócić do podglądu formy z menu kontekstowego, wybieramy *View as Form*.

Kod źródłowy projektu dla platformy .NET

Teraz wczytajmy kod źródłowy projektu, który w poprzednim rozdziale przygotowaliśmy dla platformy *.NET*, żeby przekonać się, jak wiele jest w nim różnic w porównaniu do projektu dla *Win32*. Wczytywanie projektu odbywa się identycznie jak w przypadku projektu dla *Win32*, z tą jedną różnicą, że Delphi musi być uruchomione w trybie, który pozwala na edycję projektów *.NET*.

Ponownie zobaczymy plik *Kolory.dpr*. Okazuje się, że różni się on od poznanego wcześniej w kilku znaczących szczegółach. Przede wszystkim rzuca się w oczy zbiór dyrektyw, które znajdują się bezpośrednio za linią program *Kolory*; (rysunek 2.2). Odpowiadają one za zawartość gałęzi *References* widocznej w oknie projektu; wskazuje ona ścieżki dostępu do zarządzanych bibliotek wykorzystywanych w projekcie. Poza bibliotekami platformy *.NET* (np. *System.dll*) mogą tam być biblioteki zawierające komponenty *VCL.NET* i inne klasy związane z Delphi (pliki *Borland.Vcl.dll*, *Borland.Delphi.dll* i *Borland.VclRtl.dll*).

Dalsza część kodu różni się jeszcze dwoma rzeczami. Po pierwsze za dyrektywą preprocesora wczytującą plik zasobów znajduje się linia z napisem *Program/Assembly Information*. Jeżeli klikniemy znak plusa widoczny na lewym marginesie przy tej linii (rysunek 2.2), to... pewnie westchniemy oszołomieni. Pojawi



Rysunek 2.2. Plik Kolory.dpr w projekcie dla platformy .NET

się bowiem znaczna ilość kodu i komentarzy. Nie ma tam jednak żadnej linii typowej dla Pascala, a wszystkie linie otoczone są nawiasami kwadratowymi. Są to atrybuty przechowujące numer wersji, nazwę producenta, znaki zastrzeżone i inne informacje, które ustaliliśmy w oknie opcji projektu. W przypadku projektu Win32 wszystkie te informacje umieszczone zostały w pliku zasobów *Kolory.res*. Tym razem ukryte zostały w bloku *Program/Assembly Information*, który można na szczęście zwinąć z powrotem do jednej linii. Wówczas zobaczymy ponownie blok *begin..end*, a w nim polecenia inicjujące aplikację. Te polecenia są już identyczne, jak w projekcie Win32. Przed tym blokiem dodany jest jednak atrybut *STAThread*. Mówiąc w wielkim skrócie, informuje on platformę .NET, że nasza aplikacja powinna komunikować się z systemem w trybie pojedynczego wątku (proszę nie przejmować się, jeżeli to Czytelnikowi nic nie mówi).

Natomiast jeżeli zajrzemy do pliku modułu *Unit1.pas*, który powinien nas interesować najbardziej, to okaże się, że poza nieco większą liczbą zadeklarowanych w sekcji *uses* modułów nie różni się on niczym od swojego odpowiednika dla platformy Win32.