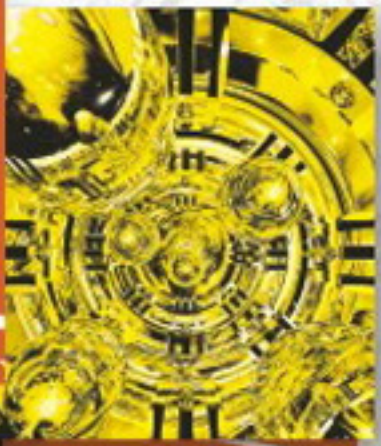
АНДРЕЙ БОРОВСКИЙ

bhv

ПРОГРАММИРОВАНИЕ В DELPHI 2005



- ОБЗОР НОВШЕСТВ DELPHI 2005 IDE
- ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ НА ПЛАТФОРМЕ WINDOWS 2000/XP/2003
- СЕКРЕТЫ СОЗДАНИЯ ПРИЛОЖЕНИЙ ADO.NET
- МНОГОУРОВНЕВЫЕ ПРИЛОЖЕНИЯ, КОМПОНЕНТНОЕ ПРОГРАММИРОВАНИЕ
- ПРИМЕРЫ НАПИСАНИЯ ГРАФИЧЕСКИХ И МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Андрей Боровский

ПРОГРАММИРОВАНИЕ В DELPHI 2005

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92Delphi2005
ББК 32.973.26-018.2
Б83

Боровский А. Н.

Б83 Программирование в Delphi 2005. — СПб.: БХВ-Петербург, 2005. — 448 с.: ил.

ISBN 5-94157-409-6

Книга посвящена разработке в Delphi 2005 различных типов приложений для Windows 2000/XP/2003. Описаны приемы программирования Win32 с учетом специфики Windows 2000/XP/2003, архитектура .NET и особенности создания приложений Windows Forms и VCL.Forms. Рассмотрены разработка приложений bdExpress, WebSnap и WebBroker, а также интернет-приложений с использованием компонентов Internet Direct 10. Уделено внимание многоуровневому компонентному программированию и бизнес-ориентированному моделированию с помощью компонентов ECO.

Описаны технологии ADO.NET, Borland Data Provider, ASP.NET и разработка приложений баз данных с помощью ADO.NET и ASP.NET. Рассмотрено создание мультимедиа-приложений с использованием расширенных возможностей графики GDI+, а также .NET и DirectX 9 SDK.

Для программистов

УДК 681.3.068+800.92Delphi2005
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.03.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 36,12.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-409-6

Оглавление

Предисловие

Книга, которую вы держите в руках, предназначена для опытных программистов Delphi. Но строгого определения понятия "опытный программист" не существует. При написании этой книги автор считал опытным программистом Delphi любого, кто хотя бы несколько лет программировал в одной (или нескольких) версиях Delphi. Говорят, что писать книги для начинающих программистов легко. Автор написал такую книгу, и знает, что это не так. Но и писать книги для опытных программистов тоже непросто. Главная трудность заключается в подборе материала. Чего еще не знает опытный программист? Что он хотел бы узнать? Задача упрощается, когда пишешь о каком-то практически совершенно новом продукте, каким был, например, Delphi 8. Сталкиваясь с новым продуктом, каждый, в известном смысле, оказывается в положении новичка. В такой ситуации у меня, по крайней мере, не возникает вопрос, о чем нужно писать. Но Delphi 2005 отличается тем, что сочетает в себе новаторство и традицию. Новаторство заключается в объединении в одной среде разработки разных языков программирования, предназначенных, к тому же, для разных платформ (Win32 и .NET). Объем технологий, охваченных Delphi 2005, превышает объем технологий, включенных в любую другую среду разработки от Borland (о чем можно судить хотя бы по размеру дистрибутива). Среда разработчика тоже стала более удобной. Лично я нашел в ней много такого, что давно уже хотел видеть в комфортной среде IDE. Традиционность же Delphi 2005 заключается в том, что это по-прежнему старая добрая среда Delphi и почти все программы, написанные для предыдущих версий пакета, будут компилироваться и в Delphi 2005.

Таким образом, всякий, кто хочет описывать Delphi 2005, сталкивается с обилием старых и новых технологий, многие из которых, если рассматривать их досконально, заслуживают отдельной книги. К тому же технологии эти охватывают совершенно разные отрасли программирования и очень сильно различаются в концептуальном плане.

Учитывая все выше сказанное, автор решил не связывать основную идею книги с какой-то технологией программирования (если не считать само

программирование в Delphi 2005). Основная идея книги заключается в другом. Авторов книг, посвященных различным программным продуктам, часто, и иногда не без основания, упрекают в "списывании" со справочных систем этих продуктов. И хотя, по моему мнению, даже такие книги могут быть полезны (учитывая, что справочные системы обычно написаны по-английски, а английским владеют не все), основная цель этой книги как раз в том и заключается, чтобы рассказать о том, чего нет в справочной системе Delphi 2005. Естественно, невозможно написать книгу о Delphi 2005, которая бы вообще не пересекалась со справочной системой Delphi 2005. Но можно написать книгу, дополняющую и расширяющую сведения справочной системы, заполняющую пробелы и (насколько возможно) приводящую более наглядные примеры.

Информация, предоставляемая справочной системой, может быть дополнена в трех аспектах. Во-первых, справочная система — это все-таки справочник. Иногда в ней не хватает общего обзора той или иной технологии. Во-вторых, иногда справочная система содержит неполные или непонятные объяснения, например, разделы справочной системы Delphi 2005, посвященные созданию служб Windows 2000+, по моему мнению, могут скорее запутать читателя, чем помочь разобраться. То же самое можно сказать об описании механизма наследования неименованных каналов в документации MSDN. Я вовсе не хочу сказать, что в моей книге не может быть таких же, или еще более серьезных, "ляпов". Каждый человек делает ошибки, и каждый имеет право исправлять ошибки других, если знает — как. Ну и наконец, справочная система не может охватить весь материал, необходимый программисту. Тем более опытному. В книге автор попытался (и надеется, что это ему удалось) рассказать о том, что должно быть наиболее интересно программистам Delphi, уже освоившим эту систему и знакомым с содержанием context help (контекстной справки).

В общем, если набор знаний программиста можно сравнить с неким программным обеспечением, то данную книгу следует рассматривать как "патч" для этого ПО или, лучше сказать, "набор патчей".

Всем серьезным программистам, работающим в Delphi, приходится читать дополнительную литературу по различным технологиям программирования, написанную с расчетом на другие языки программирования. Данная книга снабжена списком подобной литературы. В главах, посвященных разным технологиям, даются ссылки на соответствующие книги из этого списка. Список литературы не очень длинный, но, по мнению автора, все перечисленные в нем источники являются классикой своего жанра. В список добавлено несколько интернет-сайтов, на которых программисты Delphi могут найти полезную для себя информацию.

Несколько слов нужно сказать о прилагаемом компакт-диске. Диск содержит полные исходные тексты примеров программ, описанных в книге. В книге содержатся только авторские примеры, и читатель имеет полное

право делать с этими текстами все, что ему (читателю) заблагорассудится (это право, естественно, ограничено правами издательства, прежде всего на компакт-диск в целом). Структура диска очень проста. Каждый каталог с именем Ch*XX* содержит примеры программ для главы *XX*. Ссылки на примеры в книге выглядят так: если в *главе 9* вы встречаете ссылку типа "полные исходные тексты этой программы можно найти в каталоге EventHandlers", значит, на компакт-диске эти тексты находятся в каталоге Ch09\EventHandlers. Компакт-диск содержит примеры не всех программ, обсуждаемых в книге. Если какого-то примера на диске нет, это вызвано одной из двух причин: либо текст примера слишком тривиален для размещения специальной программы на диске, либо примеры программ требуют наличия дополнительных файлов (например, объектов ActiveX сторонних приложений), на распространение которых у автора нет прав (у читателя, законно владеющего соответствующими приложениями, есть право *писать* такие программы, но для их *распространения* могут потребоваться дополнительные права).

В заключение автор хотел бы сказать, что он будет рад узнать мнение читателей о своей книге, ознакомиться с замечаниями и даже постарается ответить на вопросы, связанные с материалом книги. Для связи с автором вы можете использовать адрес электронной почты: **borovsky@pochtamt.ru**.

ГЛАВА 1

Новое в языке программирования Delphi

Полное описание новшеств языка программирования Delphi 2005 неразумно размещать в одной главе. Прежде всего потому, что, в отличие от предыдущих версий Delphi, Delphi 2005 — это фактически не одна, а три среды программирования "в одном флаконе", так что сравнивать Delphi 2005 с предыдущими версиями следует сразу по трем направлениям. Для начала, необходимо сравнить Delphi 2005 для Win32 с Delphi 7 (Delphi 8 не обладала самостоятельной средой программирования для Win32). Delphi 2005 для Win32 отличается от Delphi 7 гораздо существеннее, чем Delphi 7 от Delphi 6. Далее, сопоставляя Delphi 7 и Delphi 2005 для .NET мы, фактически, сравниваем две разных, хотя и совместимых, системы разработки. Но и сравнивая среду программирования .NET в Delphi 2005 и Delphi 8 мы найдем немало различий. Наконец, сравнивая Delphi 7 и Delphi 8 с одной стороны и среду программирования Delphi 2005 для C# с другой стороны, мы должны рассматривать средства разработки, ориентированные на разные языки программирования (так, как если бы мы сравнивали Delphi и C++ Builder).

Таким образом, эта глава отражает то новое, что появилось в Delphi 8 по сравнению с Delphi 7, а также добавления, сделанные в Delphi 2005 с точки зрения языка программирования Delphi Language. Следующая глава посвящена новшествам интегрированной среды разработки. Описания программирования в Delphi 2005 на языке C# приводится в *главе 3*. В результате первые две главы дают обзор новшеств Delphi 2005, которого может быть достаточно для программистов, не собирающихся использовать C#. *Глава 6*, посвященная C#, является кратким обзором этого языка программирования, предназначенного, в основном, для "перевода" исходных текстов с языка C# на Delphi Language. Впрочем, *главу 6* рекомендуется прочитать всем программистам, которые собираются осваивать .NET.

Новшества в Delphi Language

Описывая новые элементы языка Delphi Language и среды разработки Delphi 2005, за "точку отсчета" мы возьмем Delphi 7 (а не непосредственного предшественника Delphi 2005 — Delphi 8). Выбор для сравнения именно Delphi 7 обоснован двумя причинами. Во-первых, Delphi 2005 можно рассматривать как непосредственное продолжение Delphi 7 (Delphi 8 таким продолжением не была), а значит, многие программисты, особенно те, кто программирует для Win32, перейдут на Delphi 2005 с Delphi 7. Во-вторых, Delphi 2005 появилась менее чем год спустя после выхода Delphi 8. Это означает, что даже среди тех программистов, которые начали работать с Delphi 8 и программировать для .NET, не все еще овладели новшествами языка Delphi Language образца Delphi 8.

Новая модель идентификаторов

Для того чтобы понять новые особенности Delphi 2005 (и Delphi 8), следует запомнить, что среда разработки должна теперь подчиняться неким правилам, общим для всех средств, ориентированных на платформу .NET. Многие элементы языка программирования определяются теперь не стандартами Object Pascal или Delphi Language и даже не стандартами ОС Windows. В частности, это касается идентификаторов.

Имена многих новых идентификаторов типов в Delphi 2005 совпадают с ключевыми словами языка программирования или с идентификаторами, которые в предыдущих версиях использовались в совершенно ином контексте. Для того чтобы избежать конфликта идентификаторов, в Delphi 2005 введена новая схема обозначения идентификаторов. Рассмотрим, например, класс `Type`, являющийся частью .NET Framework. В Delphi 8 этот класс определен в модуле `System`. Само слово `Type` является зарезервированным в языке Delphi Language, поэтому для объявления переменной соответствующего типа следует воспользоваться одним из двух вариантов:

```
MyVar : System.Type;
```

или

```
MyVar : &Type;
```

Первый вариант является традиционным для Delphi методом разрешения конфликтов идентификаторов — указывается полное имя идентификатора, включающее имя модуля, в котором этот идентификатор объявлен. Второй вариант введен в Delphi 8 и представляет собой сокращение первого. Префикс `&` указывает компилятору, что следующий за ним описатель является идентификатором, а не зарезервированным словом.

Пространства имен

В соответствии со структурой общей среды выполнения .NET в Delphi 8 введено понятие *"пространство имен"*. В рамках языка Delphi Language пространство имен можно рассматривать как дополнение концепции модуля. Каждый модуль декларирует собственное пространство имен. Важнейшим отличием системы пространств имен от традиционной системы модулей является возможность создавать иерархии пространств имен. Система иерархических пространств имен в Delphi Language служит той же цели, что и аналогичные системы в других языках программирования — она позволяет избежать конфликтов, возникающих при совпадении имен идентификаторов. Кроме того, соблюдение иерархии пространств имен требуется средой .NET, т. к. в ней все идентификаторы включают пространства имен.

Рассмотрим все вышесказанное на простом примере. Что мы делаем при программировании на традиционном Delphi Language, когда два модуля, используемых нашим модулем, содержат объекты с одинаковыми именами? Для того чтобы различать такие объекты, мы добавляем имя модуля к имени объекта, например, `System.Close`. Концепция пространств имен развивает этот подход. Теперь пространство имен `System` содержит кроме традиционных объектов модуля `System` ряд дочерних пространств имен.

Мы можем объявить переменную:

```
Navigator : System.Xml.XPath.XPathNavigator;
```

Это означает, что переменная `Navigator` имеет тип `XPathNavigator`, принадлежащий пространству имен `XPath`, являющемуся дочерним пространством имен по отношению к пространству имен `Xml`, которое, в свою очередь, принадлежит пространству имен `System`. Для того чтобы такое объявление переменной было возможным, раздел `uses` модуля должен содержать пространство имен `System.Xml.XPath`.

На практике пространства имен реализуются следующим образом. В каталоге, содержащем модули и пакеты библиотеки времени выполнения, можно найти файл System.Xml.dcpil. Этот файл, являющийся пакетом, содержит модули, реализующие дочерние пространства имен пространства имен `System.Xml`. Модуль, в котором определяются элементы, принадлежащие пространству имен `System.Xml.XPath`, должен иметь имя `System.Xml.XPath` (речь идет именно об имени модуля, т. к. имя файла, в котором хранится данный модуль, должно включать еще и соответствующее расширение). Таким образом, имя модуля содержит полный путь к реализованному модулем пространству имен. Имена иерархически подчиненных пространств имен при этом разделяются точками.

Выше уже говорилось о том, что среда .NET использует полные идентификаторы пространств имен для обращения к объектам. По этой причине

иерархия пространств имен Delphi 8 соответствует иерархии пространств имен .NET. В Delphi 2005 концепция пространства имен подверглась дополнительной переработке. Первое отличие: понятие пространства имен распространено теперь и на среду программирования для Win32. Поскольку пространства имен хорошо согласуются с концепцией модулей, работающие в среде Win32 программисты могут и не заметить различий, за исключением введения новой терминологии в справочной системе и подсказках в интегрированной среде разработки.

Второе отличие более существенно. В Delphi 2005 появилась возможность агрегации нескольких модулей в одном пространстве имен. Это означает, что модуль, использующий несколько других модулей, может рассматривать идентификаторы, определенные в этих модулях, как принадлежащие одному пространству имен.

Пусть, например, у нас есть два модуля: `Unit1` и `Unit2`, которые мы хотим использовать в модуле `Unit3`. В таком случае мы можем написать в разделе `uses` модуля `Unit3`:

```
uses
My.New.Namespace in 'unit1.pas;unit2.pas';
```

Теперь все идентификаторы, определенные в модулях `Unit1` и `Unit2`, с точки зрения модуля `Unit3` будут принадлежать пространству имен `My.New.Namespace`. Поскольку в каждом данном пространстве имен не может быть двух одинаковых идентификаторов, идентификаторы в модулях `Unit1` и `Unit2` не должны повторяться, иначе компилятор выдаст сообщение об ошибке.

Новые типы данных

Прежде чем рассказать о новых типах данных, следует отметить, что и старые типы данных изменились в новых версиях Delphi. В среде программирования, предназначенной для .NET, типы данных были приведены в соответствие требованиям .NET Framework. Тем, кто привык программировать на Delphi, новая система типов данных может показаться не такой уж и новой. Отличительной чертой Delphi является стремление использовать для переменных-объектов динамически выделяемую память. Классы библиотеки компонентов VCL и их наследники ориентированы на использование именно динамической памяти и этим они отличаются от простых типов, таких как `Integer` или `Char`. Среда .NET заимствовала многие архитектурные особенности Delphi, в том числе и различие типов данных. Среда .NET делит типы переменных на размерные и ссылочные. К *размерным* относятся простые типы, содержащие атомарные значения. Переменные размерных типов во многом похожи на обычные переменные языка Delphi Language: их инициализация выполняется с помощью оператора присваивания, а не с по-

мощью вызова конструктора, и даже до инициализации эти переменные имеют значения, которые хотя и являются бессмысленными с точки зрения программы, позволяют корректно работать с этими переменными. Высвобождение памяти для переменных размерных типов также происходит несколько иначе, чем для переменных ссылочных типов. Кроме описанных выше простых типов к размерным типам относятся также типы-записи (объявленные с помощью ключевого слова `record`).

Ссылочные типы представляют собой указатели на объекты классов, хранящихся в куче (heap). Для инициализации переменных ссылочных типов необходимо вызывать конструкторы. Поскольку до вызова конструктора переменная ссылочного типа содержит значение `nil`, работать с этой переменной до ее инициализации нельзя. Переменные ссылочных типов не уничтожаются сразу, как только выходят из области видимости. Высвобождение занимаемой ими памяти выполняется сборщиком мусора по мере необходимости, и эта память вообще может не высвобождаться до конца работы программы. Для гарантированного высвобождения критических ресурсов в ссылочных типах реализован *механизм финализации*, который отсутствует в размерных типах. Подробнее об управлении памятью в .NET Framework говорится в *главе 7*.

Отношение между размерными и ссылочными типами может показаться более простым, чем оно есть на самом деле. Хотя переменные размерных типов и похожи на "обычные" переменные, между ними все же существуют различия. Переменные размерных типов являются объектами и у них есть методы.

Рассмотрим простейший пример переменной размерного типа:

```
B : Byte;
```

Тип `Byte` позволяет хранить в переменной целочисленные значения от 0 до 255, и для его инициализации не нужен конструктор, однако, в соответствии с моделью .NET, у типа `Byte` есть методы. Например, выражение

```
B.ToString
```

возвращает строковое представление значения переменной `B`.

Еще один метод типа `Byte` — метод `Parse`, выполняющий обратное преобразование из строки в число (листинг 1.1).

Листинг 1.1. Преобразование типов `Byte` и `String`

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S : String;
  B : Byte
```

```
begin
  S := '6';
  B := B.Parse(S);
  Label1.Caption := B.ToString;
end;
```

Обратите внимание, что новое значение не присваивается экземпляру `Byte`, а возвращается как значение функции. Простой вызов `B.Parse(S);` не приведет к изменению значения переменной `B`. Вот почему переменная `B` встречается слева от оператора присваивания.

Следует также помнить, что все размерные типы приводимы к типу `System.TObject`, который является предком всех ссылочных типов. Более подробно различия между размерными и ссылочными типами среды .NET описаны в книге [7].

Многие размерные типы данных Delphi Language и .NET, будучи аналогичными по сути, имеют разные имена (размерные типы .NET определены в пространстве имен `System`). Для того чтобы, с одной стороны, сохранить обратную совместимость на уровне кода, а с другой стороны, облегчить использование .NET Framework, в Delphi для .NET введены типы-"двойники" традиционных типов. Каждому размерному типу .NET соответствует тип Delphi для .NET (табл. 1.1).

Таблица 1.1. *Соответствие типов Delphi Language и .NET*

Тип Delphi Language	Описание	Тип .NET
`Integer`	Знаковый 32-битный	`Int32`
`Cardinal`	Беззнаковый 32-битный	`UInt32`
`Shortint`	Знаковый 8-битный	`SByte`
`Smallint`	Знаковый 16-битный	`Int16`
`Longint`	Знаковый 32-битный	`Int32`
`Int64`	Знаковый 64-битный	`Int64`
`Byte`	Беззнаковый 8-битный	`Byte`
`Word`	Беззнаковый 16-битный	`UInt16`
`Longword`	Беззнаковый 32-битный	`UInt32`

Кроме этого в Delphi для .NET определен тип `UInt64` — беззнаковый 64-битный тип, соответствующий одноименному типу .NET.

Сложнее обстоит дело с логическими типами. Все логические типы (`Boolean`, `ByteBool`, `WordBool` и `LongBool`), определенные в прежних версиях Delphi, сохранились и в новой версии. Сложности возникают при переводе

типов. В .NET существует тип Boolean, синонимом которого является ключевое слово C# bool. В Delphi 8 тип Boolean занимает 1 байт, а тип Bool эквивалентен типу LongBool и занимает 4 байта. При этом Delphi-тип Boolean автоматически конвертируется в .NET-тип Boolean.

Тип Char в .NET — 16-битный, что соответствует типу WideChar в Delphi, однако традиционный тип Delphi Char (занимающий 1 байт) автоматически преобразуется в тип Char .NET.

Типы Real и Extended в Delphi для .NET эквивалентны типу Double. Типы Real48 и Comp не поддерживаются при программировании для .NET. Delphi-типы Single и Double соответствуют одноименным типам .NET. Тип Currency теперь основан на .NET-типе Decimal, который также определен в Delphi для .NET.

Переменные всех перечисленных выше типов имеют методы, подобные методам типа Byte. С методами можно работать и при манипуляции значениями соответствующих типов. Например, строка

```
Label1.Caption := Sin(Pi/4).ToString;
```

эквивалентна строке

```
Label1.Caption := FloatToStr(Sin(Pi/4));
```

Тип данных ShortString сохранен в Delphi 8 исключительно ради обратной совместимости. В .NET существует класс String, который оперирует двухбайтовыми символами и потому в большей степени соответствует типу Delphi 8 WideString, нежели типу AnsiString. Тип String в Delphi 8 соответствует классу String в .NET. Тип PChar в Delphi для .NET совпадает с типом PWideChar.

Работа со строками

Для работы со строками в Delphi для .NET реализован тип String, очень похожий на одноименный тип традиционной среды для Win32. Особенность типа String заключается в том, что, будучи ссылочным типом, он допускает инициализацию с помощью операции присваивания.

Класс String содержит множество полезных методов для обработки строк. Одни из этих методов имеют аналоги в наборе функций для работы со строками, реализованные в предыдущих версиях Delphi. Другие методы предоставляют новые возможности. Рассмотрим несколько примеров использования класса String. Методы ToUpper и ToLower позволяют изменять регистр символов строки (ToUpper переводит все символы строки в верхний регистр, ToLower — в нижний). Очевидно, что для правильного выполнения подобной операции у метода должна быть информация об используемой кодировке. Методы ToUpper и ToLower перегружены, т. е. каждый из них су-

ществует в двух вариантах — без параметра и с параметром CultureInfo, передающим информацию о том, как должны преобразовываться символы. При вызове методов без параметра необходима информация о языках и кодировках символов, предоставляемая операционной системой. Вызов методов с параметром CultureInfo позволяет обрабатывать строки, содержащие символы кодировок, не установленных в данной системе. Рассмотрим пример использования параметра CultureInfo (листинг 1.2).

Листинг 1.2. Пример использования класса CultureInfo

```
var
  S : String;
begin
  S := 'здравствуй, мир';
  Label1.Caption := S.ToUpper(CultureInfo.Create($0419));
end;
```

Сначала мы присваиваем переменной S строку 'здравствуй, мир', содержащую символы русского алфавита в нижнем регистре. Свойству Label1.Caption будет присвоена строка, в которой все символы переведены в верхний регистр. Для этого вызывается метод ToUpper, которому передается параметр CultureInfo, содержащий данные, необходимые для правильной обработки символов русского алфавита. При вызове метода ToUpper с таким параметром коды символов будут преобразованы правильно даже в нерусифицированной системе.

Параметр CultureInfo представляет собой класс, определенный в пространстве имен System.Globalization. Обратите внимание на то, что экземпляр класса создается "на месте вызова". Это новая характерная особенность, связанная со спецификой программирования для .NET. Динамически созданные классы не уничтожаются явным образом, поэтому если экземпляр класса нужен исключительно как параметр какого-либо метода, его можно создать прямо в строке вызова метода. Система .NET сама позаботится об уничтожении экземпляра класса, когда он не будет нужен (подробнее об этом будет сказано в следующих главах).

У класса CultureInfo несколько конструкторов. Параметры этих конструкторов позволяют указать используемый набор правил — *"культуру"* в терминологии .NET, а также, следует ли программе пользоваться настройками системы. В приведенном выше примере используемая "культура" указывалась с помощью численного идентификатора. Другой вариант конструктора CultureInfo позволяет применять для этой же цели строковый идентификатор. Например, численному значению $0419 соответствует строка 'ru-RU'. Полный перечень идентификаторов культур можно найти в справочной системе Delphi 8.

Несмотря на все изменения, которые претерпел тип `String`, к нему по-прежнему можно применять старые функции и приемы работы (листинг 1.3).

Листинг 1.3. Приемы работы с типом `String`

```
var
  S : String;
begin
  SetLength(S, 1024);
  S[3] := 'a';
end;
```

Тип `String` по-прежнему можно использовать как динамический массив.

Новые конструкции языка

В следующих разделах мы рассмотрим новые конструкции языка Delphi Language. Сначала будут описаны новые конструкции Delphi 2005, затем — конструкции, появившиеся еще в Delphi 8.

Новые конструкции Delphi Language в Delphi 2005

Рассматриваемые здесь конструкции применимы к средам .NET и Win32.

Цикл *for in do*

В Delphi 2005 появился новый вариант цикла `for`, упрощающий перебор значений из заданного диапазона. Например, следующий фрагмент программы (листинг 1.4) приведет к распечатыванию на консоли всех заглавных букв латинского алфавита.

Листинг 1.4. Пример цикла `for in do`

```
var
  i : Char;
begin
  for i in ['A'..'Z'] do Write(i);
end.
```

Оператор

```
for i in ['A'..'Z'] do
```

эквивалентен

```
for i := 'A' to 'Z' do
```

Если A — переменная-массив элементов некоторого типа, а i — переменная того же типа, то оператор

```
for i in A do
```

выполнит перебор всех элементов массива A с помощью i. Удобство этого оператора заключается в том, что нам не нужно заботиться об указании нижней и верхней границ массива. Границы будут учтены автоматически. С другой стороны, чрезмерное увлечение этой формой оператора может сделать ваш код трудночитаемым.

Очень удобно применять цикл for in do при переборе элементов класса Collection, например:

```
Item : ListViewItem;
...
for Item in ListView1.Items do ...
```

Примечание

Оператор for in do подобен оператору C# foreach, но не соответствует ему полностью. В частности, цикл for in do нельзя применять в тех случаях, когда для перебора значений используются итераторы .NET.

Встраиваемые процедуры и функции

Процесс вызова процедуры или функции занимает определенное количество машинного времени. В ситуации, когда быстрота работы программы важнее чем компактность ее кода, многие языки программирования позволяют использовать *встраиваемые функции*. Встраиваемые функции и процедуры не вызываются программой. Вместо этого их код просто добавляется в участок вызова. Таким образом, мы выигрываем в скорости (не тратится время на вызов функции), но проигрываем в размерах программы (код функции помещается в программу много раз). Теперь встраиваемые функции (и процедуры) появились и в Delphi Language. Для того чтобы определить встраиваемую функцию или процедуру, к ее заголовку необходимо добавить ключевое слово inline (листинг 1.5).

Листинг 1.5. Определение встраиваемой функции

```
function Sin2x(x : Double) : Double; inline;
begin
  Result := 2*Sin(x)*Cos(x);
end;
```

Встраиваемыми могут быть не только отдельные функции и процедуры, но и методы классов. Следует помнить, что компилятор не всегда делает функ-

цию или процедуру, помеченную как `inline`, встраиваемой. Во-первых, встраивание функций не всегда приводит к реальной оптимизации работы программы. Во-вторых, для встраиваемых процедур и функций существует ряд ограничений. Функция (процедура) не может быть встраиваемой, если она:

- является конструктором или деструктором класса;
- является методом класса и помечена как виртуальная, динамическая или как обработчик сообщения;
- является методом класса и обращается к другому методу с более ограниченной областью видимости;
- содержит ассемблерную вставку;
- объявлена в разделе модуля `interface` и обращается к элементам, объявленным в разделе `implementation`;
- вызывается как часть условного выражения операторов `while...do` и `repeat...until` (но та же функция может быть встраиваемой при вызове в других местах программы).

Кроме описанных выше, существуют и другие, менее распространенные случаи, когда функция или процедура не может быть встроенной.

Новые символы в идентификаторах

Delphi 2005 позволяет использовать в идентификаторах символы Unicode, в том числе символы кириллицы, так что теперь в программах на Delphi можно писать, например так, как указано в листинге 1.6.

Листинг 1.6. Русские буквы в именах переменных

```
var
  Ускорение, Время, Скорость, Путь : Double;
begin
  Ускорение := 9.81;
  Время := 10;
  Скорость := Ускорение*Время;
  Путь := Ускорение*Sqr(Время)/2;
  Write('Скорость = ', Скорость, 'Путь = ', Путь);
end.
```

Стоит отметить, что среда .NET умеет перекодировать регистр нелатинских букв, так что в соответствии с правилами Delphi Language идентификатор `Время` тождественен идентификатору `время`. В Delphi 2005 кириллицу можно использовать не только в проектах .NET, но и в проектах Win32.

Примечание

Хотя мне, как автору книги про Delphi, было бы удобнее использовать кириллицу для обозначения функций и переменных (раскладку клавиатуры пришлось бы переключать гораздо реже), я все же считаю, что программы с такими переменными теряют читабельность и привычный вид, и не советую злоупотреблять этой возможностью.

Многомерные динамические массивы

Еще одно новшество Delphi 2005 — возможность объявления многомерных массивов с определением длины во время выполнения программы. Рассмотрим пример использования динамических двумерных массивов (листинг 1.7).

Листинг 1.7. Пример умножения матриц

```
program TestMatrixMultiplication

{$APPTYPE CONSOLE}

uses
  SysUtils;

type
  TMatrFloat = array of array of Double;

(* Процедура умножения двух матриц R = M1*M2
   процедура сама определяет размерность массива R *)
procedure MultMatrix(const M1, M2 : TMatrFloat; var R : TMatrFloat);
var
  i, j, k : Integer;
begin
  if Length(M1[0]) <> Length(M2) then
  raise Exception.Create('Число столбцов M1 не равно числу строк M2');
  SetLength(R, Length(M1));
  for i := 0 to Length(M1) - 1 do SetLength(R[i], Length(M2[0]));
  for i := 0 to Length(M1) - 1 do
  for j := 0 to Length(M2[0]) - 1 do
  begin
    R[i, j] := 0;
    for k := 0 to Length(M1[0]) - 1 do
    R[i, j] := R[i, j] + M1[i, k]*M2[k, j];
  end;
end;
```

```
// Вывод матрицы на печать
procedure PrintMatrix(const M : TMatrFloat);
var
  i, j : Integer;
begin
  for i := 0 to Length(M)-1 do
  begin
    for j := 0 to Length(M[i])-1 do
    Write(M[i, j], ' ');
    WriteLn;
  end;
  WriteLn;
end;

var
  i : Integer;
  A, B, C : TMatrFloat;
begin
  SetLength(A, 3);
  SetLength(A[0], 2);
  SetLength(A[1], 2);
  SetLength(A[2], 2);
  A[0, 0] := 1; A[0, 1] := 2;
  A[1, 0] := 6; A[1, 1] := 10;
  A[2, 0] := 4; A[2, 1] := 11;
  SetLength(B, 2);
  SetLength(B[0], 2);
  SetLength(B[1], 2);
  B[0, 0] := 3; B[0, 1] := 2;
  B[1, 0] := 4; B[1, 1] := 1;
  MultMatrix(A, B, C);
  PrintMatrix(A);
  PrintMatrix(B);
  PrintMatrix(C);
end.
```

В этом примере мы определяем динамический двумерный массив элементов типа `Double` с помощью конструкции

```
array of array of Double
```

Для удобства мы присваиваем новому типу имя `TMatrFloat`. Первое, что должна сделать процедура умножения матриц — определить размерность каждой матрицы. Выяснить размерность двумерного массива `M` по первому индексу можно с помощью вызова функции `Length(M)`. Для второго индекса

можно воспользоваться значением длины первой строки массива M: Length(M[0]). Однако тут надо учесть один нюанс. Строки динамического многомерного массива не обязательно должны иметь одну и ту же длину (обратите внимание, что в тексте программы длина каждой строки матриц A и B задается отдельно). Это означает, что матрица типа TMatrFloat может быть и не прямоугольной. В принципе в процедурах MultMatrix и PrintMatrix мы должны были бы проверять длину всех строк матриц-аргументов. Мы не делаем этого только для упрощения листинга.

Примечание

Реализация динамических многомерных массивов в Delphi 2005 делает их похожими на массивы-указатели C#, что упрощает "перевод" программ из C# в Delphi Language.

Наконец, обращаем ваше внимание на то, что для выделения многострочного комментария были использовали символы (* и *). Этот способ выделения комментариев появился в языке Pascal очень давно и возможен до сих пор. Многие программисты считают, что лучше использовать для этой цели символы { и }. Однако, по мнению автора, в среде, интегрированной с C#, имеет смысл отказаться от такого способа выделения комментариев, чтобы не создавать путаницу (в C# символы { и } имеют совсем другой смысл).

Раз уж речь зашла о комментариях, стоит отметить новый тип комментариев .NET (появившийся еще в Delphi 8). Это комментарии, начинающиеся символами ///. От обычных эти комментарии отличаются тем, что их можно использовать при автоматической генерации документации к проекту (текст комментариев добавляется в этом случае в файл документации).

Новые элементы, введенные в Delphi 8

Поскольку среда Delphi 8 была ориентирована исключительно на .NET, новшества, введенные в язык Delphi Language в Delphi 8, продиктованы, в основном, требованиями .NET Framework. Все эти новшества сохранились и в Delphi 2005, хотя за пределами среды программирования для .NET они и не приносят особой пользы.

Новые определители видимости элементов классов

В соответствии с общей языковой спецификацией .NET (.NET Common Language Specification) в язык Delphi Language, в дополнение к уже имеющимся, введены новые ключевые слова, определяющие видимость полей и методов класса. Эти ключевые слова — strict private и strict protected.

Элементы класса, объявленные в разделе `strict private`, видимы только для методов данного класса. В отличие от элементов, объявленных в разделе `private`, эти элементы невидимы для других процедур и функций из того же модуля.

Элементы класса, объявленные в разделе `strict protected`, видимы только для методов данного класса и для методов его потомков (независимо от того, в каком модуле объявлены классы-потомки). Так же, как и элементы класса, объявленные в разделе `strict private`, элементы класса, объявленные в разделе `strict protected`, невидимы для других процедур и функций из того же модуля.

Примечание

Ключевое слово `strict` является таковым только внутри объявления класса. За пределами объявления класса это слово может использоваться как идентификатор.

В Delphi 8 введено ключевое слово `static`, позволяющее объявлять статические методы классов. Смысл статических элементов классов такой же, как и в языке C++ (и C#), т. е. статические методы используются независимо от конкретных экземпляров класса — им не передается неявный параметр `Self`. В листинге 1.8 приводится пример объявления статического метода в классе и его вызова.

Листинг 1.8. Объявление и вызов статического метода

```
TMyClass = class
  ...
public
  function MyFunc : Integer; static;
  ...
end;
...
i := TMyClass.MyFunc;
```

В силу своей специфики статические методы класса не могут напрямую обращаться к элементам класса, зависящим от экземпляра класса (т. к. для такого обращения требуется неявный параметр `Self`). Статический метод класса может работать с нестатическими методами своего класса, только если у него есть явная ссылка на экземпляр такого класса.

Статические методы классов можно использовать для определения свойств классов, и тогда эти свойства допустимо вызывать так же, как и статические методы. Для объявления таких свойств следует использовать сочетание `class property` (листинг 1.9).

Листинг 1.9. Объявление и вызов статического свойства класса

```
TMyClass = class
strict private
  class var
  FX: Integer;      // переменная для хранения значения свойства X
strict protected
  function GetX: Integer; static;       // выполняет: Result := FX
  procedure SetX(val: Integer); static; // выполняет: FX := val;
public
  class property X: Integer read GetX write SetX;
end;
...
TMyClass.X := 123;
```

Обратите внимание, что переменная, хранящая значение "статического" свойства, должна быть объявлена как `class var` (к таким переменным могут напрямую обращаться статические методы классов).

Декларация новых типов внутри классов

Платформа .NET позволяет объявлять новые типы внутри объявлений классов, и Delphi поддерживает такую возможность. В листинге 1.10 показано, как объявить один класс внутри другого, как создавать экземпляры внешнего и внутреннего классов и как получать доступ к их методам.

Листинг 1.10. Объявление класса внутри другого класса

```
type

  TGarage = class
  public
    type
      TCar = class
      strict private
        FColor : Integer;
      public
        constructor Create(Color : Integer); override;
        function GetColor : Integer;
      end;
      procedure AddCar(Car : TCar);
  private
     FCar : TCar;
  end;
...
```

```
constructor TGarage.TCar.Create;
begin
  inherited Create();
  FColor := Color;
end;

function TGarage.TCar.GetColor;
begin
  Result := FColor;
end;

procedure TGarage.AddCar;
begin
  FCar := Car;
end;
...
procedure TForm1.FormCreate(Sender: TObject);
var
  Garage : TGarage;
  Car : TGarage.TCar;
  Color : Integer;
begin
  Garage := TGarage.Create;
  Car := TGarage.TCar.Create($ff0000)
  Garage.AddCar(Car);
  Color := Car.GetColor;
end;
```

Объявление нового типа внутри класса производится с помощью ключевого слова `type`, так же как объявление нового типа в модуле. После этого внешний класс может использовать переменные нового типа.

Описывая методы внутреннего класса, в заголовке метода следует сначала указать имя внешнего класса, а затем имя внутреннего класса, отделенное от имени внешнего класса точкой. Таким же образом следует указывать внутренний класс и при объявлении переменной соответствующего типа. Поскольку для полной идентификации внутреннего типа требуется указывать и имя внешнего типа, ничто не мешает нам определить внешний тип (класс) с таким же именем, как у какого-либо внутреннего типа, но с другим содержанием. Нет также причин, запрещающих нам объявить внутренний тип с таким же именем в каком-либо другом классе.

Декларация констант внутри классов

Delphi 8 позволяет объявлять внутри класса не только типы, но и константы (листинг 1.11).

Листинг 1.11. Объявление констант внутри класса

```
type

  TMyClass = class
  public
    const
      Hello = 'Hello';
      HelloWorld = Hello + ' World!';
    ...
  end;
...
WriteLn(TMyClass.HelloWorld);
```

Новые типы классов

В Delphi 8 были введены новые спецификаторы типов классов: `abstract` и `sealed`. Эти спецификаторы противоположны по своему смыслу. Если класс объявлен как `abstract` (абстрактный), это означает, что данный класс не может использоваться в программах непосредственно. Для того чтобы использовать функциональность абстрактного класса, необходимо определить класс-потомок этого класса (даже в том случае, если в самом абстрактном классе нет методов, помеченных как `abstract`).

Если класс объявлен как `sealed` (закрытый), на его основе нельзя создавать классы потомков. Данное требование касается всех языков программирования, поддерживаемых .NET. Это значит, что если вы опишете закрытый класс, другие программисты не смогут создавать классы-потомки указанного класса, даже если они пишут на другом языке программирования (например, C#). Точно так же и вы не можете создавать потомков закрытого класса, независимо от того, был ли этот класс написан на Delphi Language или на каком-либо другом языке. В листинге 1.12 приводится пример объявления закрытого класса.

Листинг 1.12. Объявление закрытого класса

```
TMyClass = class sealed
  private
    FX : Integer;
  public
    procedure SetX(X : Integer);
    function GetX : Integer;
  end;
```