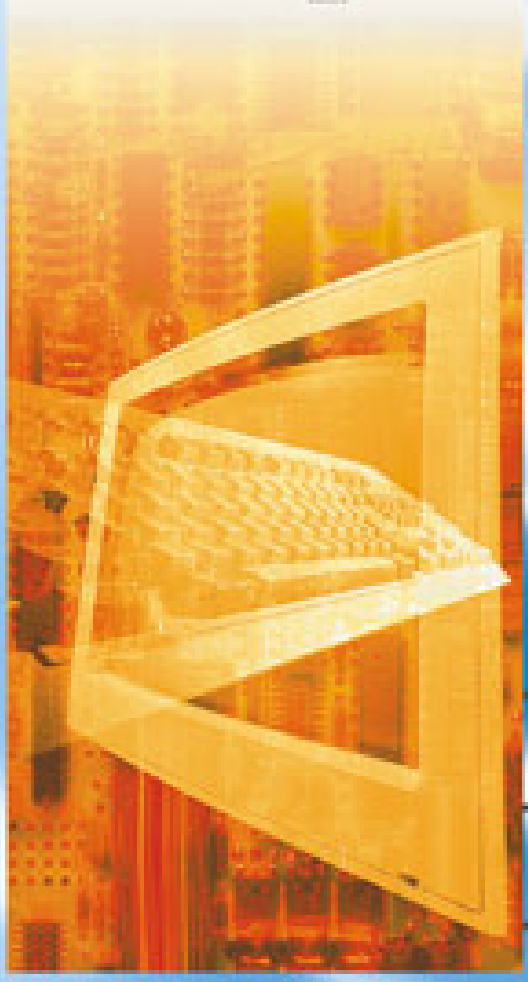
Евгений Марков
Владимир Никифоров

bhv

Delphi 2005 для .NET



- Архитектура Microsoft .NET
- Языки программирования Delphi и C#
- Windows Forms
- VCL.NET
- ADO.NET, DBP.NET
- ECO

Наиболее полное руководство

В ПОДЛИННИКЕ®

Евгений Марков
Владимир Никифоров

Delphi 2005 для .NET

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92Delphi2005
ББК 32.973.26-018.1
М27

Марков Е. П., Никифоров В. В.

М27 Delphi 2005 для .NET. — СПб.: БХВ-Петербург, 2005. — 896 с.: ил.

ISBN 5-94157-701-X

Рассмотрены практические аспекты программирования в Borland Delphi 2005 для .NET. Описаны вопросы реализации .NET в Delphi, а также синтаксис и объектные модели двух языков программирования — Delphi и C#. Показаны особенности разработки приложений для двух основных типов приложений — Windows Forms и VCL.NET. Большое внимание уделено созданию приложения для работы с базами данных. Рассмотрены технологии ADO.NET и BDP.NET, а также создание приложений VCL.NET, поддерживающих известные технологии доступа к данным — Borland DataBase Engine .NET, dbExpress .NET, InterBase Express .NET, dbGo и др. Описаны приемы создания приложений на основе технологии ECO, использующей перспективную архитектуру разработки приложений MDA. Излагаемый материал сопровождается примерами.

Для подготовленных программистов

УДК 681.3.068+800.92Delphi2005
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.07.05.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 72,24.
Тираж 3000 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 ОАО "Техническая книга"
190005, Санкт-Петербург, Измайловский пр., 29

ISBN 5-94157-701-X

Оглавление

ГЛАВА 3

Язык программирования Delphi

Начиная знакомство с Borland Developer Studio и следующей версией языка Delphi — очередным этапом развития столь мощного объектно-ориентированного языка, — прежде всего необходимо обратить внимание на саму идеологию объектно-ориентированного программирования (ООП). Этот экскурс вооружит читателя терминами, столь часто употребляемыми нами в последующих главах.

Объектно-ориентированное программирование

Вот уже как минимум десять лет программисты Delphi живут в терминах ООП и все это время теория ООП изменяется, подстраиваясь под нужды разработчиков программного обеспечения, меняя, однако, своей сути, в основе которой три краеугольных камня:

- инкапсуляция;
- наследование;
- полиморфизм.

Данные понятия неотделимы от двух других, принятых в Delphi и C#, — класса и объекта. Рассмотрим эти сущности, опираясь на синтаксис языка Delphi (отличия, характерные для C#, будут описаны в *главе 4*).

Классы и объекты

Классом в Delphi называется структура языка, которая может иметь в своем составе переменные, функции и процедуры. Переменные, в зависимости от предназначения, называются *полями* или *свойствами*. Процедуры и функции класса называются *методами*. Соответствующий классу тип будем называть *объектным типом* или *объектом*:

```
type
  TMyObject = class(TObject)
    MyField: Integer;
    function MyMethod: Integer;
end;
```

В этом примере описан класс `TMyObject`, содержащий поле `MyField` и метод `MyMethod`.

Существует также возможность внутри описания класса объявить не только свойство и/или метод, но и класс (так называемый вложенный тип данных):

```
type
  className = class [abstract | sealed] (ancestorType)
      memberList
      type
         nestedTypeDeclaration
      memberList
end;
```

Следующий пример (листинг 3.1) демонстрирует описание и доступ к полям и методам вложенного класса.

Листинг 3.1. Организация доступа к полям и методам вложенного класса

```
type
   TOuterClass = class
      strict private
      myField: Integer;
      public
         type
            TInnerClass = class
              public
                myInnerField: Integer;
                procedure innerProc;
            end;
         procedure outerProc;
   end;
```

Реализацию метода `innerProc` теперь можно осуществить с помощью следующей конструкции:

```
procedure TOuterClass.TInnerClass.innerProc;
begin
   ...
end;
```

Поля объекта аналогичны полям записи (record). Это данные, уникальные для каждого созданного в программе экземпляра класса. Описанный здесь класс `TMyObject` имеет одно поле — `MyField`.

Методы — это процедуры и функции, описанные внутри класса и предназначенные для операций над его полями. В состав класса входит указатель на специальную таблицу, где содержится вся информация, нужная для вызова методов. От обычных процедур и функций методы отличаются тем, что им при вызове передается указатель на тот объект, который их вызвал. Поэтому обрабатываться будут поля именно этого объекта. Внутри метода указатель на вызвавший его объект доступен под зарезервированным именем `Self`.

Понятие свойства будет подробно рассмотрено далее. Пока можно определить его как поле, доступное для чтения и записи не напрямую, а через соответствующие методы.

Классы могут быть описаны либо в секции интерфейса модуля, либо на верхнем уровне вложенности секции реализации. Не допускается описание классов "где попало", т. е. внутри процедур и других блоков кода.

Опережающее объявление классов разрешено, что иллюстрирует листинг 3.2.

Листинг 3.2. Пример опережающего объявления классов

```
type
  TFirstObject = class;
  TSecondObject = class(TObject)
  F1st : TFirstObject;
  ...
  end;
  TFirstObject = class(TObject)
  ...
  end;
```

Чтобы использовать класс в программе, нужно, как минимум, объявить переменную этого типа. Переменная объектного типа называется *экземпляром класса*, или *объектом*:

```
var
  AMyObject: TMyObject;
```

В приведенном фрагменте кода переменная `AMyObject` на самом деле является указателем, содержащим адрес объекта, а сам объект создается с помощью специального метода — *конструктора* объекта.

```
AMyObject := TMyObject.Create;
```

Созданный экземпляр класса уничтожается другим методом — *деструктором*:

```
AMyObject.Destroy;
```

Но в целях проверки указателя на область памяти, где размещен экземпляр класса на отличие указателя от `Nil`, рекомендуется вызвать метод `AMyObject.Free`, который автоматически вызовет метод уничтожения объекта — `Destroy`.

Поля, свойства и методы

Поля являются переменными, объявленными внутри класса. Они предназначены для хранения данных во время работы экземпляра класса (объекта). Ограничений на тип полей в классе не предусмотрено. В описании класса поля должны предшествовать методам и свойствам. Обычно поля служат для обеспечения выполнения операций внутри класса.

Разработка серьезного приложения в Delphi подразумевает использование нескольких классов, разделенных в соответствии с логикой разрабатываемого приложения. Несомненно, встает вопрос о возможности взаимодействия классов друг с другом или с программными элементами приложения. Для таких целей служат свойства, описываемые с помощью зарезервированного слова `property`.

Свойства представляют собой атрибуты, наиболее целостно описывающие объект. Например, обычная кнопка, размещенная на форме приложения, обладает такими свойствами, как цвет (`Color`), размеры (`Width`, `Height`), положение (`Left`, `Top`), надпись на кнопке (`Caption`) и т. п.

Так как свойство обеспечивает обмен данными с внешней средой, то для доступа к его значению необходимы специальные методы класса. Поэтому обычно свойство определяется тремя элементами: полем и двумя методами, которые осуществляют его чтение/запись:

```
type
 TAnObject = class(TObject)
  function GetColor: TSomeType;
  procedure SetColor(ANewValue: TSomeType);
  property AColor: TSomeType read GetColor write SetColor;
 end;
```

В данном примере доступ к значению свойства `AColor` осуществляется через вызовы методов `GetColor` и `SetColor`. Однако в обращении к этим методам в явном виде нет необходимости, достаточно написать

```
AnObject.AColor := AValue;
AVariable := AnObject.AColor;
```

и компилятор самостоятельно оттранслирует обращение к свойству в вызовы методов. Таким образом, внешне свойство выглядит в точности как обычное поле, но за всяким обращением к нему могут стоять нужные вам действия.

События

Все, что происходит с вашим приложением в процессе работы под управлением операционной системы Windows, — это последовательность наступления тех или иных событий. В Delphi предусмотрен механизм отслеживания и управления подобными событиями.

Событие — это свойство процедурного типа, предназначенное для создания пользовательской реакции на то или иное входное воздействие:

```
property OnMyEvent: TMyEvent read FOnMyEvent write FOnMyEvent;
```

Здесь `FOnMyEvent` — поле процедурного типа, содержащее адрес некоторого метода. Присвоить такому свойству значение — значит указать объекту адрес метода, который будет вызываться в момент наступления события. Такие методы называют обработчиками событий. Например, запись

```
Application.OnActivate := MyActivatingMethod;
```

означает, что при активизации объекта `Application` (так называется объект, соответствующий работающему приложению) будет вызван метод-обработчик `MyActivatingMethod`.

Внутри библиотеки времени выполнения (RTL) Delphi вызовы обработчиков событий находятся в методах, обрабатывающих сообщения Windows. Выполнив необходимые действия, этот метод проверяет, известен ли адрес обработчика, и, если это так, вызывает его:

```
if Assigned(FOnMyEvent) then FOnMyEvent(Self);
```

События имеют разные число и тип параметров в зависимости от происхождения и предназначения. Общим для всех является параметр `Sender`, он указывает на объект-источник события. Самый простой тип — `TNotifyEvent` — не имеет других параметров:

```
TNotifyEvent = procedure (Sender: TObject) of object;
```

Тип метода, предназначенный для извещения о нажатии клавиши, предусматривает передачу программисту кода этой клавиши, о передвижении мыши — ее текущих координат и т. п. Все события в Delphi принято предварять префиксом `On`: `OnKeyPress`, `OnMouseMove` и т. п. Например, на форме `Form1` расположен компонент отображения статичного текста `TLabel1`. Тогда для обработки щелчка мышью (событие `OnClick`) будет создана заготовка метода `TForm1.Label1Click`:

```
procedure TForm1.Label1Click(Sender: TObject);
begin
...
end;
```

Поскольку события являются свойствами объекта, их значения можно изменять в любой момент во время выполнения программы. Эта замечательная возможность называется делегированием. Можно в любой момент взять способы реакции на события у одного объекта и присвоить (делегировать) их другому:

```
Object1.OnMouseMove := Object2.OnMouseMove;
```

Принцип делегирования позволяет избежать трудоемкого процесса порождения новых дочерних классов для каждого специфического случая, заменяя его простой подстановкой процедур. При необходимости можно выбирать один из нескольких возможных вариантов обработчиков событий.

Инкапсуляция

Как правило, объект — это сложная конструкция, свойства и поведение составных частей которой находятся во взаимодействии. К примеру, если мы моделируем взлетающий самолет, то после набора им определенной скорости и отрыва от земли принципы управления им полностью изменяются. Поэтому в объекте "самолет" явно недостаточно просто увеличить значение поля "скорость" на несколько километров в час, такое изменение должно автоматически повлечь за собой целый ряд других.

При создании программных объектов подобные ситуации можно моделировать, связывая со свойствами необходимые методы. Понятие инкапсуляции соответствует этому механизму.

Классическое правило объектно-ориентированного программирования утверждает, что для обеспечения надежности нежелателен прямой доступ к полям объекта, чтение и обновление их содержимого должно выполняться вызовом соответствующих методов. Это правило и называется инкапсуляцией.

Наследование

Второй краеугольный камень ООП — наследование. Этот простой принцип означает, что если вы хотите создать новый класс, лишь немного отличающийся от старого, то совершенно нет необходимости в переписывании заново уже существующих полей и методов. Вы объявляете, что новый класс `TNewObject`:

```
TNewObject = class(TOldObject);
```

является *потомком* или *дочерним классом* старого класса `TOldObject`, называемого *предком* или *родительским классом*, и добавляете к нему новые поля, методы и свойства, — иными словами, все то, что нужно при переходе от общего к частному.

В Delphi все классы являются потомками базового класса `TObject`. Поэтому если вы создаете дочерний класс прямо от `TObject`, то в определении его можно не упоминать. Следующие два выражения одинаково верны:

```
TMyObject = class(TObject);
TMyObject = class;
```

Первый вариант, хотя он и более длинный, предпочтительнее — для устранения возможных неоднозначностей.

Унаследованные от класса-предка поля и методы доступны в дочернем классе; если имеет место совпадение имен методов, то говорят, что они перекрываются.

Полиморфизм

Рассмотрим внимательно следующий пример (листинг 3.3). Пусть у нас имеется некое обобщенное поле для хранения данных — класс `TField` и три его потомка — для хранения строк, целых и вещественных чисел.

Листинг 3.3. Пример реализации принципа полиморфизма

```
type
 TField = class
  function GetData:string; virtual; abstract;
 end;
 TStringField = class(TField)
  FData : string;
  function GetData: string; override;
 end;
 TIntegerField = class(TField)
  FData : Integer;
  function GetData: string;override;
 end;
 TExtendedField = class(TField)
  FData : Extended;
  function GetData: string;override;
 end;
...
```