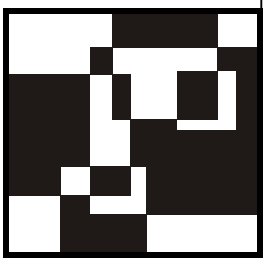
ГЛАВА 10



Программирование в Word

10.1. Зачем программировать в Word

Word — старейшее и самое популярное приложение, входящее в состав Microsoft Office. В большинстве организаций пользователи готовят документы именно в Word.

С точки зрения программирования Word — это, прежде всего, средство для изготовления отчетов к базам данных. При этом отчет — это любой документ, который формируется на основе информации из базы данных, например: договор, акт приемки-передачи, приходный кассовый ордер, объявление на взнос наличными, распоряжение в бухгалтерию, накладная и т. п. Конечно же, к отчетам, которые можно формировать в Word, относятся и документы со сводными данными — отчеты за период, ведомости и т. п.

Автору приходилось создавать приложения с отчетами, разработанными в самых разных программных продуктах — Microsoft Access, Crystal Reports, Microsoft Reporting Services и т. д. Если ваше приложение генерирует отчеты к базам данных в Microsoft Word, то, скорее всего, такие отчеты будут не самыми быстрыми, с точки зрения их формирования, и не самыми простыми, с точки зрения программирования. Зато совершенно точно они будут самыми дружелюбными по отношению к конечному пользователю. Почему?

Очень часто на предприятии возникает необходимость исправить в форме отчета всего пару строк — например, вместо "Директор" поставить "И. О. Директора". Если отчет создан в Crystal Reports или в Microsoft Reporting Services, придется срочно обращаться к разработчику. А через какое-то время И. О. утвердят в должности директора, и разработчику придется править отчет снова.

Если же отчет изначально создается в документе Word, то пользователь всегда может сам внести в созданный документ необходимые изменения — по-

давляющее большинство пользователей на предприятии умеют работать в Word. Срочно разыскивать разработчика уже не нужно.

У Word есть и другие преимущества. Как правило, при изготовлении отчетов в Word значения из базы данных подставляются в шаблон отчета, который хранится в базе данных или в файле (с расширением dot). Если формат отчета сложный, с большим количеством специфического оформления (например, объявление на взнос наличными), то намного проще подготовить его шаблон в Word, чем, к примеру, в Crystal Reports или Reporting Services.

Еще одно программное применение Word — умение работать с разными форматами документов. Эту возможность Word вполне можно использовать для массовой обработки документов.

Приведу случай из практики: в каталоге на диске у нас собралось несколько сотен "разнокалиберных" документов разных пользователей. Часть из них создана в Word разных версий, часть — просто текстовые файлы, некоторые документы в форматах HTML, XML или EML (сообщения электронной почты). На предприятии внедрена система документооборота на основе SharePoint Portal Server и нам необходимо привести все эти документы к единому формату (Word 2003) и загрузить их на SharePoint Portal Server. Конечно же, без автоматизации в такой ситуации возиться придется очень долго.

Третье программное применение Word — форматирование документов, например: программное применение стилей, поиск и замена участков текста сразу во многих документах, работа со структурой документа и т. п. Обычно такие задачи ставятся в издательствах, например, при подготовке рукописей.

10.2. Введение в программирование в Word. Объектная модель

Общая структура объектов Word выглядит так, как показано на рис. 10.1.

Но пугаться не стоит — большая часть из этих сотен объектов никогда вам не понадобится. На практике для решения большинства программных задач достаточно знать всего лишь пять объектов (с сопутствующими коллекциями):

- объект `Application`;
- объект `Document` (с коллекцией `Documents`);
- объект `Selection`;
- объект `Range`;
- объект `Bookmark` (с коллекцией `Bookmarks`).

Далее все эти важные объекты будут подробно описаны. Для каждого объекта вначале будут рассмотрены общие моменты, связанные с ними, например,

в каких ситуациях они нужны и как с их помощью выполнять те или иные действия. Поскольку наиболее часто встречающаяся задача программирования в Word — это создание документа (на основе шаблона) и запись в нужное место документа необходимой информации, то акцент будет сделан на использовании соответствующих объектов для решения именно этой задачи.

Кроме того, для каждого объекта будут перечислены самые важные свойства, методы и события с кратким их описанием. Эта часть добавлена по просьбе слушателей учебного курса по программированию в Office, поскольку многие из них не владеют английским настолько, чтобы свободно пользоваться документацией. Если вы читаете эту книгу подряд, то эти справочные части можно просто пропускать.

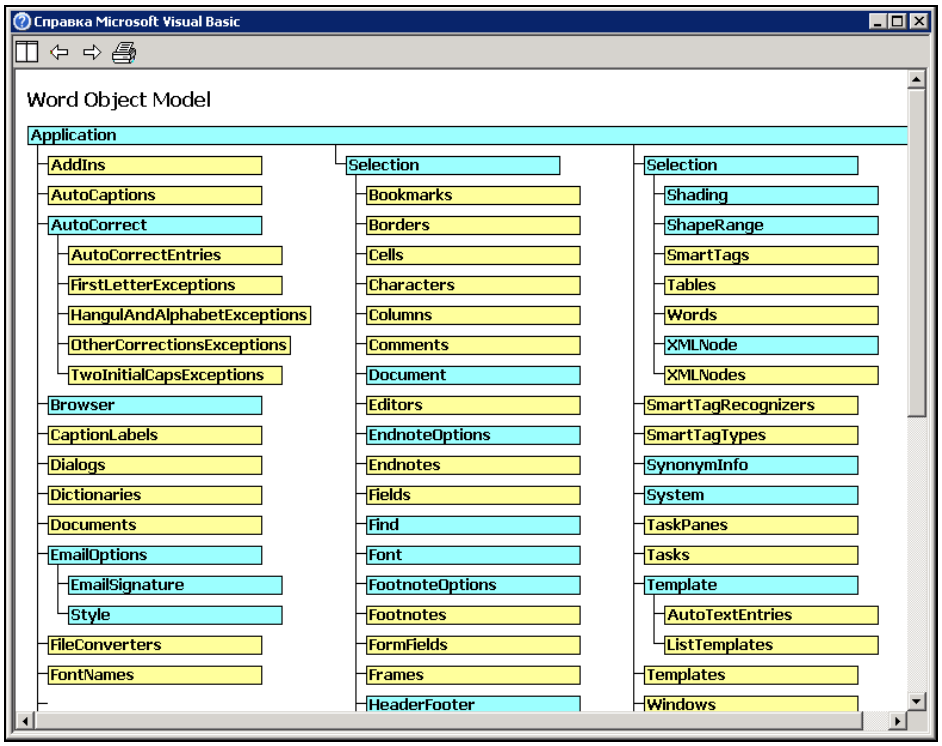


Рис. 10.1. Главные объекты Word

10.3. Объект *Application*

10.3.1. Как работать с объектом *Application*

Объект `Application` — это само приложение Microsoft Word. Все остальные объекты Word "вложены" в него. Создать этот объект — значит, запустить

Word на вашем компьютере. Как правило, это нам и необходимо (если мы создаем документ в формате Word из другого приложения, например из Access). Но не забудьте — если вы запускаете Word из другого приложения Office, то необходимо добавить в ваш проект ссылку на библиотеку Microsoft Word 11.0 Object Library.

Код запуска Word очень прост:

```
Dim oWord As New Word.Application
```

Однако, выполнив его из другого приложения, вы, скорее всего, даже не заметите, что у вас что-то произошло. Причины просты:

1. По умолчанию Word запускается в скрытом окне.
2. Если в нем не открыт ни один документ, он тут же закрывается (после того, как завершается создавшая его процедура).

Сделать Word видимым очень просто:

```
oWord.Visible = True
```

Однако может возникнуть вопрос: а нужно ли его делать видимым? Некоторые разработчики утверждают, что не нужно. Пусть Word работает в скрытом окне, создавая требуемый документ. Когда пользователю потребуется, он этот документ откроет. Как поступать в конкретном случае, решать вам, но я предпочитаю, чтобы Word все-таки был видимым: во-первых, сразу видны все проблемы при создании документа, а во-вторых, пользователям почему-то очень нравится, когда у них на глазах открывается Word и начинает печатать строки, которые в противном случае пришлось бы печатать им самим.

Если вы работаете с Word в скрытом окне, не забудьте после выполнения необходимых действий его закрыть (иначе он так и останется в оперативной памяти, видимый только через **Task Manager**). Для закрытия Word нужно вызвать его метод `Quit()`.

Чтобы Word не закрывался сам собой, в нем достаточно создать новый документ. Подробно об этом будет рассказано в следующем разделе, но самый простой вариант создания нового документа Word выглядит так:

```
Dim oWord As New Word.Application
oWord.Visible = True
oWord.Documents.Add
```

Если Word уже был открыт на компьютере, то можно получить на него ссылку, например, при помощи такого кода:

```
Set oWord = GetObject(, "Word.Application")
```

Однако на практике, кроме очень специальных случаев (активизация объектов OLE), такой подход по сравнению с открытием нового экземпляра Word

ничего не дает. Наоборот, появляется дополнительный риск нечаянно испортить открытый в существующем экземпляре созданный пользователем документ или закрыть его без сохранения пользовательских документов. Поэтому лучше создавать новый экземпляр Word.

Если же ваш код VBA выполняется в Word (т. е. Word уже запущен), объект `Application` создавать не надо. В этой ситуации он будет автоматически доступен в любой момент (чтобы в этом убедиться, достаточно впечатать в окне редактора кода `Application` и добавить точку). Более того, если не указано, к какому объекту относится то или иное свойство или метод, компилятор VBA в Word автоматически считает, что это свойство или метод принадлежит объекту `Application`. Поэтому следующие два фрагмента кода функционально одинаковые:

```
Application.Selection.TypeText "Мой текст"
```

и

```
Selection.TypeText "Мой текст"
```

Еще один важный момент, который связан с объектом `Application` в Word. Для него предусмотрено большое количество удобных в использовании событий (открытие документа, выход из Word, щелчок правой кнопкой мыши, изменение документа, печать документа, сохранение документа и т. п.) Однако по умолчанию все эти события не видны. Чтобы они появились, необходимо в разделе `Declarations` кода формы (а не модуля!) объявить объект `Application` с ключевым словом `WithEvents`, например так:

```
Public WithEvents App As Word.Application
```

В списке объектов у вас появится новый объект `App` (т. е. `Application`), для которого можно выбрать события и добавлять код в событийные процедуры точно так же, как мы это делали для формы и элементов управления.

10.3.2. Свойства, методы и события объекта *Application*

Далее для справки приведены самые важные свойства, методы и события объекта `Application`.

- `ActiveDocument` — возвращает объект активного документа в данном экземпляре Word. Это свойство используется очень активно, обычно без упоминания объекта `Application`, например:

  ```
  ActiveDocument.Save
  ```

  Свойство доступно только для чтения, поэтому чтобы сделать какой-нибудь документ активным, придется вызывать для его объекта метод `Activate()`.

- `ActivePrinter` — позволяет получить или настроить активный принтер в ходе работы программы. Также используется очень активно, например, если результаты работы вашего приложения необходимо печатать на определенном сетевом принтере. Свойство доступно как для чтения, так и для записи.
- `AutomationSecurity` — определяет уровень безопасности при программном открытии файлов. По умолчанию установлено значение `msoAutomationSecurityLow` — открывать с включенными макросами. Можно также использовать значения `msoAutomationSecurityForceDisable` (отключить макросы) и `msoAutomationSecurityByUI` (то, что настроено на графическом интерфейсе).
- `BackgroundPrintingStatus` — определяет, сколько заданий Word стоит в очереди на печать.
- `Browser` — свойство, которое возвращает объект `Browser` (малозаметный набор из трех кнопок, который прячется под вертикальной полосой прокрутки). С программной точки зрения интересно его свойство `Target`, которое может принимать одно из 12 значений (комментарий, сноска, таблица, рисунок, заголовок, страница и т. п.). Затем при помощи методов `Next()` и `Previous()` для этого объекта мы можем перемещаться между этими элементами.
- `Build` — возвращает версию и номер сборки Word. Очень полезно для проверки на совместимость, если ваше приложение работает только под определенными версиями Word.
- `CapsLock` — позволяет проверить, включен ли режим CapsLock на клавиатуре. Изменить этот режим при помощи данного свойства нельзя (только для чтения), для этого есть другие средства (связанные с использованием Windows API). Аналогично работает свойство `NumLock`.
- `Caption` — позволяет заменить текст "Microsoft Word" в заголовке окна на другой, например "Мое приложение".
- `CheckLanguage` — возвращает `True`, если Word определяет в автоматическом режиме язык, на котором производится ввод текста. Если в системе установлено несколько языков ввода, то по умолчанию автоматическая проверка установлена. При помощи этого свойства можно изменить режим работы Word.
- `COMAdIns` — позволяет получить ссылку на коллекцию загруженных COM Ad ins — встраиваемых в Word приложений, построенных по технологии COM. Очень удобно использовать перед обращением к данному встраиваемому приложению.

- `CustomizationContext` — свойство, которое позволяет указать шаблон или документ, на который будут распространяться внесенные вами изменения в меню, панели инструментов и клавиатурные комбинации. Например, код:

  ```
  CustomizationContext = NormalTemplate
  ```

  говорит о том, что все изменения, которые вы будете вносить, начиная с этого момента, будут сохраняться в шаблоне Normal.dot (и, таким образом, будут применяться ко всем документам).

- `Dialogs` — возвращает коллекцию `Dialogs`, представляющую из себя все возможные диалоговые окна Word. При помощи этой "ветви" объектной модели Word вы можете открыть любое из сотен диалоговых окон Word и определить действия, которые будут предприняты при выборе пользователем тех или иных параметров в данном диалоговом окне. К сожалению, эта "ветвь" очень плохо документирована, и при использовании объектов диалоговых окон приходится заниматься самостоятельными исследованиями (при помощи макрорекордера и окна **Locals**), чтобы определить нужные свойства и их значения. По моему опыту, обычно бывает проще создать свою форму VBA, которая будет выполнять необходимые действия, чем заниматься такой исследовательской работой. Пример использования диалогового окна открытия файла может выглядеть так:

  ```
  Dim oDlg As Dialog
  Set oDlg = Application.Dialogs(wdDialogFileOpen)
  If oDlg.Display = -1 Then
      MsgBox "Вы выбрали файл: " & _
          Application.Options.DefaultFilePath(wdCurrentFolderPath) & _
          "\" & oDlg.Name
  End If
  ```

  Для диалоговых окон, которые предназначены для работы с файлами, в объекте `Application` предусмотрено отдельное свойство `FileDialog`, возвращающее одноименный объект.

- `DefaultSaveFormat` — определяет формат сохранения файлов Word по умолчанию (тот, который будет предлагаться пользователь в диалоговом окне **Save As**). Можно настроить на сохранение в формате обычного текста TXT, текста Unicode, RTF и т. п.

- `DisplayAlerts` — очень важное свойство. Оно позволяет подавить вывод ошибок и диалоговых окон при работе макросов и приложений VBA. Во многих ситуациях без него не обойтись. Особенно часто прибегать к этому свойству требуется, когда в ходе работы программы необходимо что-нибудь удалить или закрыть без сохранения.

- `DisplayAutoCompleteTips` — включает или отключает подсказки для автозавершения текста. Чаще всего необходимо отключить.
- `Documents` — самое важное свойство. Возвращает коллекцию документов. Подробнее про эту коллекцию и работу с документами читайте в *разд. 10.4*.
- `EmailOptions` — возвращает очень сложный и насыщенный свойствами объект `EmailOptions`, который используется для настройки Word как редактора почтовых сообщений Outlook.
- `EnableCancelKey` — это свойство определяет, сможет ли пользователь прервать выполнение любого макроса при нажатии клавиш <Ctrl>+<Break>. Если установить для этого свойства значение `wdCancelDisabled`, то это приведет к тому, что макрос, вошедший в бесконечный цикл, можно будет закрыть только вместе с Word — через Task Manager.
- `FeatureInstall` — еще одно свойство, которое позволяет не раздражать пользователя попытками Office доустановить еще не установленные компоненты. Для этого нужно установить это свойство в значение `msoFeatureInstallNone`.
- `FileDialog` — возвращает объект `FileDialog`, т. е. окно выбора файла, каталога, открытия файла или сохранения. Для открытия этого окна необходимо воспользоваться методом `Show()` этого объекта.
- `FileSearch` — возвращает объект `FileSearch`, который может использоваться для поиска файлов по определенным параметрам.
- `International` — еще одно очень важное свойство. Возвращает информацию о текущих региональных настройках даты, времени, валюты, отображения чисел, локализации версии Word и т. п.
- `IsObjectValid` — очень удобное свойство для всевозможных проверок (открыт ли документ, находится ли указатель в таблице и т. п.). Проверяет, существует ли еще объект, к которому мы хотим обратиться. Позволяет уберечь от ошибок, когда, например, документ или объект в документе был удален пользователем.
- `KeyBindings` — очень удобное во многих ситуациях свойство. Оно возвращает коллекцию `KeyBindings` — привязок клавиатурных комбинаций. Проще говоря, при помощи этого объекта и его подобъектов вы можете назначить любую команду Word или любой макрос любому сочетанию клавиш (в том числе и тем, которые уже заняты служебными командами, например <Alt>+<F4>). Общая последовательность действий при этом выглядит так:

- определяем свойство `CustomizationContext` объекта `Application`, т. е. где будут сохраняться наши изменения: в шаблоне Normal.dot, в текущем документе или в шаблоне, прикрепленном к текущему документу;
- при помощи метода `Application.BuildKeyCode()` определяем цифровой код для нашей клавиатурной комбинации;
- при помощи метода `KeyBindings.Add()` добавляем новое назначение, при этом определяем все необходимые параметры.

Например, чтобы по нажатию клавиш <Alt>+<D> у нас запускался макрос `DataLoad()` во всех документах, можно выполнить следующий код:

```
CustomizationContext = NormalTemplate
Application.KeyBindings.Add wdKeyCategoryMacro, _
    "Normal.NewMacros.DataLoad", BuildKeyCode(wdKeyAlt, wdKeyD)
```

- `Language` — еще одно свойство, которое позволяет определить, установлена ли на компьютере пользователя локализованная версия Word (точнее, это свойство определяет язык пользовательского интерфейса). Для русского языка будет возвращаться значение 1049, для английского — 1033. Более подробную информацию (о языке помощи, языке программы установки и т. п.) можно получить при помощи свойства `LanguageSettings`.
- `MacroContainer` — очень полезное свойство для программистов. Позволяет в ходе выполнения макроса определить, откуда был запущен текущий программный код (обычно проверяются два варианта — Normal.dot или текущий документ).
- `NewDocument` — одна из возможностей создать новый документ Word. Возвращает объект `NewDocument`. Для создания нового документа используется метод `Application.NewDocument.Add()`.
- `NormalTemplate` — это свойство позволяет получить ссылку на объект `Template`, представляющий Normal.dot, для внесения в него изменений.
- `Option` — возвращает объект `Option` с огромным количеством свойств. Через этот объект программным способом можно настроить значения на всех вкладках окна **Параметры** (меню **Сервис | Параметры**).
- `Path` — возвращает путь к программным файлам Word на диске.
- `PrintPreview` — с помощью этого свойства можно перейти в режим предварительного просмотра текущего документа или проверить, находимся ли мы в этом режиме. Очень удобно для показа документа пользователю или для реализации своей процедуры печати.
- `ScreenUpdating` — свойство, которое позволяет запретить перерисовку экрана (если установить его значение в `False`). Обычно используется для ускорения работы процедур, которые выводят что-то на экран.

- `Selection` — еще одно важнейшее свойство. Возвращает объект `Selection` — то место, в котором находится указатель вставки. Подробнее о нем — в *разд. 10.5*.
- `ShowStartupDialog` — определяет, показывать или нет **Task Panel** (панель задач в правой части документа) при запуске Word. Чаще всего используется для отключения показа. Есть еще несколько свойств с префиксом `Show...`, значения которых очевидны.
- `SpecialMode` — позволяет проверить, не находится ли Word в специальном режиме копирования и вставки (для перехода в этот режим нужно выделить текст и нажать <F2> или <Shift>+<F2>, а потом переместить курсор и нажать <Enter>).
- `StartUpPath` — предоставляет возможность просмотреть/определить путь к каталогу автозапуска. Те шаблоны и встраиваемые приложения, которые находятся в этом каталоге, Word при запуске открывает автоматически. По умолчанию каталог автозапуска находится в профиле пользователя. Путь к нему выглядит как **\application data\microsoft\word\startup**.
- `StatusBar` — еще одно очень полезное свойство. Позволяет вывести текст в Status Bar (строка состояния), т. е. в строке в нижней части окна приложения, где выводится информация о страницах, столбцах, языке, режимах работы и т. п.
- `System` — возвращает одноименный объект `System`, предназначенный для получения информации из операционной системы (региональный настройки, тип курсора мыши, разрешение экрана, тип процессора и т. п.). Позволяет также подключать сетевые диски и запускать приложение Microsoft System Information.
- `Tasks` — возвращает одноименную коллекцию `Tasks` с объектами `Task`, представляющими все работающие в системе процессы. При помощи этих объектов можно программным способом найти работающее в системе приложение и что-нибудь с ним сделать (сделать видимым или невидимым, активизировать, закрыть, передать в его окно сообщение Windows, как при работе с Windows API и т. п.). Опытные разработчики активно используют этот набор объектов для работы с внешними приложениями. Запускать внешние приложения лучше всего при помощи специального объекта `Shell`, о котором будет рассказано в *разд. 10.6.10*.
- `UserControl` — очень важное свойство (оно есть и в Excel). Это свойство позволяет определить, как именно был запущен Word — пользователем вручную или программным образом. На основе этого можно, например, сделать вывод, нужно ли его программным образом закрывать.

- `UserInitials` и `UserName` — позволяет получить или определить информацию об инициалах или имени пользователя. Инициалы используются в исправлениях, а имя пользователя — в свойствах документа.
- `VBE` — это свойство возвращает недокументированный, но очень интересный объект `VBE`, представляющий редактор Visual Basic. Обычно используется для программного внесения изменений в проекты VBA, например, добавление ссылок.
- `Version` — свойство возвращает версию Word (менее подробную, чем свойство `Build`). Для Word 2003 значение этого свойства равно 11.0.
- `Visible` — позволяет спрятать окно Microsoft Word очень качественно — Word исчезает и с рабочего стола, и из панели задач.
- `Windows` — возвращает информацию об одноименной коллекции `Windows`, содержащей объекты `Window`. Эти объекты представляют окна документов Word.
- `WindowsState` — позволяет свернуть, развернуть или восстановить окно Word.

Самые важные методы объекта `Application` приведены далее.

- `Activate()` — просто активизирует окно Word с текущим документом. Обычно нужно активизировать определенный документ, поэтому этот метод используется для объекта `Document`.
- `BuildKeyCode()` — позволяет узнать уникальный номер для клавиатурной комбинации в Word. Пример использования этого метода был приведен ранее при рассмотрении свойства `Application.KeyBindings`.
- `ChangeFileOpenDirectory()` — этот метод позволяет изменить каталог, который по умолчанию открывает Word при работе с документами (по умолчанию задан, конечно, каталог **Мои документы**);
- `CheckGrammar()` и `CheckSpelling()` — позволяют проверить грамматику и орфографию для передаваемых символьных значений. Чаще всего используются аналогичные методы для объектов `Document` и `Range`.
- `CleanString()` — очень полезный метод. Позволяет "чистить" передаваемое символьное значение (полученное, например, от объектов `Selection` или `Range`) от специальных символов Word и превращает их в обычный текст, как будто он был набран в блокноте.
- `DefaultWebOptions()` — возвращает одноименный объект, при помощи которого можно определить множество свойств, используемых при сохранении документа Word в формате HTML (кодировка, работа с изображениями, CSS, с какими браузерами обеспечивать совместимость и т. п.).

- `GoBack()` — этот метод обеспечивает переход на последнее место редактирования в документе. Word сохраняет с документом три последние точки редактирования, так что открыть последний документ в Word и перейти на точку, где вы остановились, можно очень просто:

```
RecentFiles(1).Open
Application.GoBack
```

- `GoForward()` — обеспечивает переход вперед по точкам сохранения.
- `KeyBoard()` — очень полезный метод. Позволяет программным способом переключать раскладку клавиатуры в Word, уберегая таким образом пользователей от ошибок. Переключение на русский язык выглядит так:

```
Application.Keyboard 1049
```

  а на английский:

```
Application.Keyboard 1033
```

  Если этому методу ничего не передавать, он вернет текущую раскладку клавиатуры.

- `KeyString()` — метод, обратный `BuildKeyCode()`. Если `BuildKeyCode()` возвращает уникальный идентификатор клавиатурной комбинации, то этот метод возвращает клавиатурную комбинацию для данного уникального идентификатора.
- `ListCommands()` — метод, не похожий на другие. Он создает новый документ и выводит в нем в виде таблицы справочник по методам и клавиатурным комбинациям Word, как стандартным, так и назначенным вами.
- `OnTime()` — очень интересный метод. Он позволяет выполнить макрос Word либо в указанное вами время, либо по прошествии какого-то времени. В Word одновременно может работать только один таймер. При помощи этого метода можно выполнять ресурсоемкие операции в автоматическом режиме.
- `OrganizerCopy()` — еще один полезный метод. Позволяет скопировать макрос, панель инструментов, запись автотекста или стиль из одного документа в другой. Для удаления и переименования этих элементов предусмотрены методы `OrganizerDelete()` и `OrganizerRename()`.
- `PrintOut()` — метод, который принимает огромное количество параметров (все необязательные) и позволяет вывести на печать весь документ или его часть. Может использоваться для объектов `Application`, `Document` и `Window`.
- `Quit()` — метод, который используется, видимо, чаще всех. Позволяет закрыть Word с сохранением или без сохранения документов.

- `Repeat()` — просто повторяет последнюю выполненную команду указанное вами количество раз.
- `ResetIgnoreAll()` — снять метку со всех фрагментов текста, помеченных как "без проверки" в ходе проверки орфографии.
- `Run()` — еще один очень важный метод. Позволяет запустить процедуру/макрос из открытого шаблона или документа с передачей параметров.
- `ScreenRefresh()` — обновляет окно приложения. Обычно используется после того, как автоматическое обновление было отключено при помощи свойства `ScreenUpdating`.
- `ShowClipboard()` — отображает панель буфера обмена Word (если вы работаете с несколькими буферами).

Остальные методы относятся к работе протокола DDE, преобразованию различных единиц измерений и т. п.

У объекта `Application` есть множество событий: открытие, закрытие, сохранение и печать документа, щелчки мышью, активизация, выход из приложения и т. п. Единственное, что следует еще раз отметить — события объекта `Application` по умолчанию не отображаются в редакторе Visual Basic. Чтобы они появились, в раздел `Declarations` нужно поместить следующую строку кода:

```
Public WithEvents App As Word.Application
```

В этом случае в списке объектов в окне редактора кода для форм появится объект `App` со всеми необходимыми событиями.

10.4. Коллекция *Documents* и объекты *Document*

10.4.1. Как работать с коллекцией *Documents*

На одну ступень ниже объекта `Application` в объектной модели Word (и по логике использования в приложениях) находятся коллекция `Documents` и объекты `Document`, из которых она состоит. При программировании в Word без них обычно не обойтись.

Чаще всего в программах нам нужно:

1. Запустить Word.
2. Создать или открыть документ.
3. Что-то с этим документом сделать (например, впечатать в нужные места этого документа значения, полученные из базы данных или от пользователя).

Запуск Word производится при помощи объекта `Application`, с которым вы уже знакомы. Для выполнения различных действий с документом используются объекты `Selection`, `Range` и `Bookmark`, которые будут рассмотрены в *разд. 10.5*. А вот второй пункт — создание или открытие документа, проверка, открыт уже документ или нет, сохранение документа и т. п. — реализуется при помощи коллекции `Documents` и объекта `Document`.

Самый простой вариант создания документа выглядит так:

```
Dim oDoc As Word.Document
Set oDoc = Application.Documents.Add()
```

При этом мы создали обычный пустой документ (на основе шаблона Normal.dot) и получили ссылку на него в объектную переменную `oDoc`. Далее в документ можно программно вводить нужную нам информацию.

Однако создавать пустой документ и формировать все его содержимое удобно лишь тогда, когда документ очень простой. Попробуйте программно создать какой-нибудь документ посложнее, например, объявление на взнос наличными или большой договор. Возможно, у вас это получится, но работы придется выполнить очень много. Намного проще набрать и оформить сложный документ обычным способом, как простой документ Word, и оставить в нем пустые места для заполнения из программы. Проще всего это сделать при помощи шаблона Word.

Например, в нашей ситуации нам вначале нужно набрать текст договора, оставив пустые места для изменяемых данных, и сохранить его с расширением dot (предположим, что он сохранен на диске C: с именем dog_blank.dot). Тогда создать документ на основе этого шаблона можно так:

```
Dim oDoc As Word.Document
Set oDoc = Application.Documents.Add("C:\dog_blank.dot")
```

А далее при помощи объектов `Bookmark` и `Range` *(см. разд. 10.5)* вводим текст в оставленные пустые места.

Шаблоны документов можно хранить в разных местах:

- первый вариант — просто *в файле на диске локального компьютера пользователя*. Это самый простой вариант, но у него есть недостатки: во-первых, пользователь может случайно его изменить, а во-вторых, удобнее использовать общий централизованный набор шаблонов для всех пользователей на предприятии, не копируя их на компьютер каждого пользователя;
- второй вариант — *хранить шаблоны в скрытом сетевом каталоге на файл-сервере*, доступном только на чтение. При этом нам не придется заботиться о наличии необходимого шаблона на компьютере пользователя.

Однако и есть здесь проблемы: программа получается неавтономной, зависящей от внешних файлов на файл-сервере. Перенос ее, например, в филиалы будет сопряжен со сложностями;

- третий вариант — *поместить шаблон в базу данных*. Удобнее всего поместить шаблон в виде объекта OLE в базу данных Access. В этой же базе данных Access удобно разместить код приложения, графические формы для пользователя и т. п. Запуск Word при этом будет производиться программно из Access. Конечно, такая программа сможет обращаться не только к данным в базе данных Access, но и к данным на внешних источниках: на SQL Server, Oracle и т. п.

Третий вариант наиболее удобен, поскольку ваше приложение со всеми шаблонами, программным кодом и т. п. будет состоять из единственного файла mdb, а документы будут создаваться как обычные файлы Word. Кроме того, сам по себе Access — это очень мощное приложение, в нем есть множество удобных возможностей. Единственная проблема в этом случае — то, что метод `Documents.Add()` согласен принимать шаблон только в виде файла на диске, а про базы данных он ничего не знает. Можно предварительно извлекать шаблон документа из базы данных и сохранять его в файле во временном каталоге, но лучше идти другим путем и активизировать объект шаблона в базе данных методом `OLEActivate()`. При этом у нас автоматически будет запущен Word, а в нем будет создан новый документ на основе шаблона из базы данных. Подробнее о том, как это делается — в *гл. 12*.

Часто возникает потребность программным способом не создавать новый документ, а открыть уже имеющийся и что-то сделать с ним. Открыть документ проще всего при помощи метода `Open()` коллекции `Documents`. Самый простой вариант применения этого метода выглядит так:

```
Dim oDoc1 As Word.Document
Set oDoc1 = Documents.Open("c:\doc1.doc")
```

Если документ уже открыт, то по умолчанию просто создается ссылка на этот открытый документ, вместо открытия его заново.

Важная (и неочевидная) особенность метода `Open()` заключается в том, что при его использовании во время открытия файла не открывается диалоговое окно **Предупреждение системы безопасности**, в котором пользователь может отключить макросы. За счет этого разработчик может гарантировать работоспособность своего приложения или реализовать систему защиты — от печати, доступа к разным функциям и т. п.

Сохранять документы лучше при помощи методов `Save()` и `SaveAs()` объекта `Document`. В коллекции `Documents` есть также свой метод `Save()`, который позволяет сохранить сразу все открытые документы Word, но обычно это менее удобно.

Заметим, что из Word можно открывать не только документы Word разных версий, но и документы десятков других различных форматов, про которые знает Word — TXT, HTML, XML и т. п. Сохранять файлы также можно в одном из десятков встроенных форматов или использовать свой собственный пользовательский формат (при помощи объекта `FileConvertor`). За счет этого программным образом при помощи макросов можно очень удобно преобразовывать большое количество документов, которые могут находиться, например, в разных каталогах на файловых серверах, или в общих папках Exchange Server, или в базе данных SharePoint Portal Server. Для прохода по дереву каталогов удобнее всего использовать объект `FileSystemObject` из библиотеки Microsoft Scripting Runtime, которая есть на любом компьютере Windows *(см. гл. 4)*.

10.4.2. Свойства и методы коллекции *Documents*

Коллекция `Documents`, как уже говорилось ранее, представляет все документы Word, открытые в настоящий момент. Нумерация документов в коллекции начинается с 1. Из свойств этой коллекции интерес может представлять только свойство `Count` — количество открытых документов. Гораздо важнее методы коллекции `Documents`. Про некоторые из них мы уже говорили в предыдущем разделе, но здесь для справки приведем информацию о них еще раз.

- `Add()` — этот метод позволяет создать и сразу же открыть новый документ (и вернуть ссылку на его объект). Это наиболее распространенный способ создания новых документов в Word. Полный синтаксис этого метода выглядит как:

  ```
  Add(Template, NewTemplate, DocumentType, Visible)
  ```

  Здесь `Template` — это шаблон для создания нового документа, `NewTemplate` (`True/False`) — делать ли новый документ шаблоном, `DocumentType` — тип документа, может принимать значения: `wdNewBlankDocument`, `wdNewEmailMessage`, `wdNewFrameset` или `wdNewWebPage` (по умолчанию новый чистый документ), `Visible` — будет ли новый документ видимым (по умолчанию) или невидимым. Все эти параметры являются необязательными. Если не указать ни один из них, будет просто создан новый документ на основе шаблона Normal.dot (как будто вы создали новый документ при помощи меню **Файл | Создать**).

- `Open()` — еще один важнейший метод коллекции `Documents`. Позволяет открыть документ с диска и добавить его в коллекцию. Этот метод принимает множество параметров, из которых обязательным является только один — имя документа (вместе с путем к нему). Самый простой вариант применения этого метода выглядит так:

```
Dim oDoc1 As Document
Set oDoc1 = Documents.Open("c:\doc1.doc")
```

- `Item()` — позволяет найти нужный документ в коллекции по его индексу. Обычно для получения ссылки на нужный документ используется конструкция `For...Next` с проверкой значения какого-либо свойства документа через `If...Then`. Чаще всего это свойство — `Name`:

```
Dim oDoc1 As Word.Document
For i = 1 To Documents.Count
    Set oDoc1 = Documents.Item(i)
    If oDoc1.Name = "doc1.doc" Then
        Exit For
    End If
    Set oDoc1 = Nothing
Next
```

  Этот код возвращает ссылку в виде переменной `oDoc1` на документ doc1.doc, если он есть в коллекции. Если его нет, то во избежание ошибок нужно реализовывать дополнительные проверки. На практике можно было бы перед сравнением привести имя документа в нижний регистр, если учитывать регистр букв при поиске вам не нужно.

  Через метод `Item()` можно получить доступ к объекту документа напрямую. Например, в этом примере мы получаем имя первого документа в коллекции `Documents`:

```
MsgBox Documents.Item(1).Name
```

- `Save()` и `Close()` — позволяют соответственно сохранить или закрыть все документы в коллекции.
- `CanCheckOut()` (можно ли "забрать" документ в монопольный доступ) и `CheckOut()` (забрать документ в монопольный доступ) — эти методы можно применять, если документ находится в документной библиотеке в базе данных SharePoint Portal Server.

10.4.3. Работа с объектом *Document*, его свойства и методы

После того, как мы при помощи объекта `Application` запустили Word, при помощи коллекции `Documents` создали (или открыли, или нашли среди уже открытых) нужный нам документ, можно выполнять с этим документом различные действия, реализованные при помощи свойств, методов и событий объекта `Document`. У этого объекта десятки свойств и методов, и здесь мы рассмотрим только наиболее важные и часто используемые.

Обратите внимание, что к объекту `Document` можно обращаться и не создавая специальную объектную переменную. Существует еще, по крайней мере, три способа получения доступа к объекту `Document`:

- работать с документом как с элементом коллекции `Documents`. Формат обращения может выглядеть, например, как `Documents.Item(1)`;
- использовать специальное ключевое слово `ThisDocument`. При помощи него можно получить ссылку на объект документа, которому принадлежит исполняемый программный модуль, например:

  ```
  MsgBox ThisDocument.Name
  ```

- использовать свойство `ActiveDocument` объекта `Application`. Это свойство возвращает нам объект активного документа:

  ```
  MsgBox Application.ActiveDocument.Name
  ```

  или просто

  ```
  MsgBox ActiveDocument.Name
  ```

Самые важные свойства объекта `Document` представлены далее.

- `ActiveWritingStyle` — текущий активный стиль (заголовок определенного уровня, обычный текст, гиперссылка и т. п.). Рекомендуется проверять это свойство перед вводом текста.
- `AttachedTemplate` — предоставляет возможность подключить шаблон (со всеми макросами, стилями, записями автотекста и т. п.) или проверить, какой шаблон подключен (вручную это можно сделать через меню **Сервис | Шаблоны и надстройки**).
- `Background` — возвращает объект `Shape`, представляющий фоновый рисунок (фоновые рисунки видны только в режиме Web-документ).
- `BuiltInDocumentProperties` — позволяет получить ссылку на коллекцию `DocumentProperties` с одноименными объектами, представляющими встроенные свойства документа (название, автор, категория, комментарии и т. п.);
- `Characters` — возвращает коллекцию объектов `Range`, каждый из которых представляет один символ. Это свойство есть не только у объекта `Document`, но и у объектов `Selection` и `Range`. Может использоваться, например, для выполнения операция поиска и замены или статистических подсчетов (например, если переводчику платят за количество символов);
- `Content` — свойство, возвращающее объект `Range`, представляющий собой главную цепочку документа *(main document story)*. Если говорить проще, то это просто текст документа, без колонтитулов, сносок, комментариев и т. п.

- `CustomDocumentsProperty` — свойство, возвращающее коллекцию объектов `DocumentProperties`, представляющих пользовательские свойства документа. Можно использовать для сохранения вместе с документом любых значений переменных. Очень удобно, например, для подсчета количества открытий документа, флажков печатался/не печатался, сколько раз вызывалась та или иная функция, на каких компьютерах и каким пользователем открывался и т. п.
- `DefaultTabStop` — определяет отступ по умолчанию при использовании символа табуляции. По умолчанию задано 35 пунктов, что равно примерно 1,25 см.
- `DisableFeatures` — отключает возможности, которые понимают только последние версии Word (для совместимости с пользователями, у которых на компьютерах стоят старые версии). Обычно само свойство `DisableFeatures` просто включает такой режим, а конкретный уровень совместимости задается при помощи свойства `DisableFeaturesIntroducedAfter`.
- `DoNotEmbedSystemFonts` — позволяет не вставлять в документ системные шрифты (по умолчанию вставляются для русского, японского и других языков с набором символов, отличным от латиницы). Позволяет сократить размер документа, но тогда пользователи в системе, где не установлен русский язык, не смогут прочесть этот документ.
- `EmbedTrueTypeFonts` — очень полезное свойство, если вы работаете с документом в месте, где используются экзотические шрифты (например, в издательстве). Вставка шрифтов true-type гарантирует, что получатели документа будут видеть его точно таким же, как и создатель.
- `Envelope` — позволяет получить ссылку на специальный объект `Envelope`, который используется для создания почтовых конвертов.
- `Fields` — позволяет получить ссылку на коллекцию `Fields` одноименных объектов. Это свойство очень полезно при работе с полями. Поле в Word — это место в документе, отведенное для подстановки изменяемых данных: формул, даты, информации об авторе, размере документа и т. п. При работе с документом Word средствами обычного графического интерфейса добавить новое поле можно при помощи меню **Вставка | Поле**.
- `Footnotes` — возвращает коллекцию сносок.
- Свойства с префиксом `Formatting...` — определяют, что показывать в списке стилей панели инструментов **Форматирование**.
- `FormFields` — аналогично `Fields`, но в этом случае возвращается ссылка на поля в формах.
- `FullName` — возвращает полное имя объекта (вместе с путем к нему в файловой системе или Web). Доступно только для чтения.

- `GrammarChecked` — помечает весь документ, как проверенный с точки зрения грамматики (фактически отключает проверку грамматики для данного документа). Такое же свойство существует и у объекта `Range`. Коллекцию ошибок, выловленных при проверке грамматики, можно получить при помощи свойства `GrammaticalErrors`, а выделить ошибки зеленым волнистым подчеркиванием (если этого не сделано) — при помощи свойства `ShowGrammaticalErrors`. Для орфографических ошибок существуют аналогичные свойства `SpellingChecked`, `SpellingErrors` и `ShowSpellingErrors`.
- `HasPassword` — проверяет, назначен ли пароль для указанного документа. Другое свойство `Password` назначает пароль. По причине крайней слабости защиты пароли в Word, Excel и Access использовать не рекомендуется.
- `Indexes` — возвращает коллекцию индексов (т. е. предметных указателей) для документа.
- `Name` — имя документа (без пути к нему).
- `OpenEncoding` — возвращает кодовую страницу, которая использовалась для открытия документа. Для русского языка по умолчанию это 1251.
- `PageSetup` — позволяет получить ссылку на одноименный объект. Используется в основном при реализации печати.
- `Paragraphs` — возвращает ссылку на коллекцию абзацев в данном документе.
- `Path` — возвращает путь к документу в файловой системе (без имени). Это свойство может пригодиться, чтобы создать еще один файл в том же каталоге.
- `Permission` — позволяет получить доступ к объекту `Permission`, который управляет системой внутренних разрешений документа Word (но не разрешений файловой системы).
- `PrintRevisions` — определяет печатать или нет пометки редактора (исправления) вместе с документом. По умолчанию — печатать.
- `ProtectionType` — проверяет защиту данного документа (разрешено все, или только комментарии, чтение, изменения в полях форм и т. п.). Сама защита устанавливается при помощи метода `Protect()`.
- `ReadOnly` — определяет, можно ли вносить изменения в документ или он доступен только для чтения. Это свойство само доступно только для чтения, поскольку соответствующий атрибут устанавливается в файловой системе.
- `RemoveDateAndTime` и `RemovePersonalInformation` — удаляют информацию о дате и времени произведенных изменений и всю информацию о пользова-

теле из документа (включая свойства документа). Могут быть полезными при создании файла-образца.

- `Saved` — очень важное свойство. Позволяет определить, изменялся ли документ со времени последнего сохранения.
- `SaveEncoding` — позволяет явно указать (или получить) кодировку, которая будет использоваться при сохранении документа.
- `SaveFormat` — позволяет получить информацию о формате документа (DOC, RTF, TXT, HTML и т. п.). Доступно только для чтения.
- `Sections` — возвращает коллекцию разделов документа. `Sentences` — то же самое для предложений. Аналогично работают свойства `Shapes`, `Styles`, `Subdocuments`, `Tables`, `Windows` и `Words`.
- `Type` — возвращает тип документа (обычный, шаблон или Web-страница с фреймами).
- `Variables` — еще одно очень удобное свойство. Можно использовать для сохранения своих служебных данных вместе с документом, как и пользовательские атрибуты *(custom attributes)*, но, в отличие от пользовательских атрибутов документа, пользователям эти свойства не будут видны.

Теперь приведу самые важные методы объекта `Document`.

- `Activate()` — этот метод позволяет сделать указанный вами документ активным (например, для ввода текста).
- `AddToFavorities()` — позволяет добавить ссылку на документ в каталог **Избранное**. Может быть полезным, если пользователь будет работать с этим документом постоянно.
- `CheckSpelling()` и `CheckGrammar()` — запускают проверку орфографии и грамматики соответственно.
- `Close()` — закрывает документ. Можно закрыть с сохранением (по умолчанию), а можно без (если указать соответствующий параметр).
- `Compare()` — сравнивает документ с другим и генерирует редакторские пометки в местах, где обнаружены различия.
- `DetectLanguage()` — определяет язык текста. Проверка производится по предложениям, на основе сверки слов со встроенными словарями. Такая проверка производится автоматически во время ввода текста или при открытии нового документа. Чтобы заново провести проверку языков, свойство `LanguageDetected` нужно перевести в `False`.
- `FitToPages()` — очень интересный метод. Размер шрифта автоматически меняется таким образом, чтобы текст стал занимать на одну страницу меньше. Можно использовать для устранения "висячих страниц" и других проблем верстки.

- `FollowHyperlink()` — открывает указанный вами документ в соответствующем приложении (если HTML, то в Internet Explorer).
- `GoTo()` — очень мощный метод. Для объектов `Document` и `Range` он возвращает объект `Range`, для объекта `Selection` — просто перемещает указатель ввода текста на нужное место. Возвращаемые объекты в зависимости от параметров, которые были переданы этому методу, могут указывать на начало страницы, на определенные строки, закладки, комментарии, таблицы, секции, поля, ссылки, формулы и т. п. Может переходить на определенный номер такого элемента в документе, первый, последний, следующий и т. п. Очень удобно использовать для установки указателя в нужное место для автоматического ввода текста.
- `Merge()` — позволяет произвести слияние двух документов. Метод очень сложный и мощный, основывается на применении редакторских пометок.
- `PresentIt()` — открывает данный документ в PowerPoint.
- `PrintOut()` — очень сложный метод, который позволяет вывести на печать весь документ или его часть.
- `PrintPreview()` — переводит документ в режим предварительного просмотра.
- `Protect()` — ограничивает внесения изменений в документ при помощи пароля или нового средства управления правами на доступ к данным, которое называется IRM. Те же возможности на графическом экране доступны через меню **Файл | Разрешения**.
- `Range()` — очень важный метод. Возвращает объект `Range` *(см. разд. 10.5)*, принимает в качестве параметров номер начального символа диапазона и номер конечного символа.
- `Redo()` — повторяет последнее действие. В качестве параметра принимает количество последних действий, возвращает `True`, если повтор был произведен успешно.
- `Repaginate()` — выполняет переразбивку документа на страницы. Обычно используется, если автоматическая разбивка была ранее отключена (например, на вкладке **Общие** диалогового окна **Параметры** (меню **Сервис | Параметры**) или программно при помощи объекта `Options`).
- `Save()` — смысл этого метода очевиден. Если документ ранее не сохранялся, открывается диалоговое окно **Сохранить как**.
- `SaveAs()` — очень мощный и сложный метод. Можно определить путь для сохраняемого документа, его формат, кодировку, пароли на открытие и изменение документа, вставку шрифтов и многое другое. Очень удобно использовать, например, для автоматической конвертации документов.

- `Select()` — позволяет просто выделить весь документ. Этот метод существует для очень большого количества объектов, в том числе для `Selection` и `Range`.
- `TransformDocument()` — исключительно мощный метод, но только для программистов, которые хорошо разбираются в XML и XSLT. Позволяет применить к документу таблицу преобразований стилей (Extensible Stylesheet Language Transformation, XSLT), при помощи которой можно поменять все, что угодно.
- `Undo()` — отменяет определенное количество последних действий. По синтаксису и принципам работы — полный аналог метода `Redo()`.
- `UndoClear()` — очищает буфер отмены изменений, чтобы пользователь не смог откатить произведенные действия.
- `UnProtect()` — снимает защиту с документа (определенную методом `Protect()` или в графическом интерфейсе). Может быть очень полезным перед программным внесением изменений в защищенный документ.

Часто используемых событий у объекта `Document` всего три: `New` (можно определить только для шаблона, срабатывает, когда на основе этого шаблона создается новый документ), `Open` и `Close`. Все эти события очевидны и изначально доступны в окне редактора кода Visual Basic.

10.5. Объекты *Selection*, *Range* и *Bookmark*

10.5.1. Работа с объектом *Selection*

После того как мы запустили приложение, нашли и активизировали нужный нам файл, следующее действие, которое выполняется чаще всего, — ввод или редактирование текста в нужном месте. Для этого используются объекты `Selection`, `Range` и `Bookmark`. Каждое из них используется в своих ситуациях и для своих задач.

Первый объект, который мы рассмотрим, — это объект `Selection`.

Обычно перед тем, как что-либо сделать в окне документа Word, пользователь либо выделяет нужный фрагмент текста, либо переставляет указатель вставки текста в нужное место. Объект `Selection` представляет именно такой выделенный участок текста (а если ничего не выделено пользователем, то место, где находится указатель вставки). Именно этот объект обычно использует макрорекордер.

Создавать объект `Selection` и получать на него ссылку в переменную совершенно не нужно. Дело в том, что объект `Selection` в документе может быть

только один. Он создается автоматически при запуске Word и всегда доступен. Обращаться к нему можно так:

```
Application.Selection.Text = "Вставляемый текст"
```

или просто:

```
Selection.Text = "Вставляемый текст"
```

Обычно нам нужно правильно определить то место, на которое указывает объект `Selection`, чтобы выделить нужный нам участок текста или точку для ввода.

Существует несколько способов для настройки выделения в документе Word:

- самый простой способ — положиться на выделение нужного текста пользователем. Обычно такой способ применяется для сложного редактирования или форматирования участков текста и для ввода информации в указанное пользователем место документа, когда в автоматическом режиме нужное место не найти;
- воспользоваться методом `Select()`, который предусмотрен для огромного числа объектов (`Document`, `Range`, `Bookmark`, `Table` со всеми подобъектами типа столбцов и строк, `PageNumber`, `Field` и т. п.). Этот метод просто выделяет весь документ, закладку, таблицу и т. п.;
- воспользоваться многочисленными методами объекта `Selection`, чтобы преобразовать уже существующее выделение;
- воспользоваться объектом `Find` для поиска нужного фрагмента текста. Подробнее об этом объекте — в *разд. 10.6.5*;
- если вам нужно ввести информацию в самое начало документа, можно вообще ничего не делать. По умолчанию указатель вставки устанавливается на начало документа. Только не забудьте сделать этот документ активным.

Если вы полагаетесь на выделение нужного места пользователем, то помните, что пользователь может ухитриться выделить одновременно несмежные участки текста (при помощи клавиши <Ctrl>) или выделить не текст, а часть таблицы, рисунок или другой нестандартный объект в документе. Чаще всего поведение программы, работающей с объектом `Selection`, в этом случае становится совершенно непредсказуемым, поэтому рекомендуется всегда использовать дополнительные проверки при помощи свойств `Type` и `Information` объекта `Selection`.

Несмотря на то, что применение объекта `Selection` — самый простой и наглядный метод редактирования текста, и чаще всего именно он используется макрорекордером, на практике программисты применяют его редко. Объясняется это очень просто: при использовании этого объекта мы слишком зави-

сим от действий пользователя. Если во время выполнения нашего кода пользователь проявит инициативу и начнет щелкать по документу мышью, результат может быть непредсказуемым. Защититься от вмешательства пользователя можно двумя способами:

- работать со скрытым (т. е. невидимым) документом или, возможно, со скрытым экземпляром Word. Для включения и отключения видимости можно использовать свойство `Visible` объектов `Document` или `Application`;
- более удобный способ — вместо объекта `Selection` использовать объекты `Range` и `Bookmark`, о которых будет рассказано в следующих разделах.

10.5.2. Свойства и методы объекта *Selection*

Вначале расскажем о самых часто используемых свойствах объекта `Selection`.

- `Bookmarks` — возвращает коллекцию `Bookmarks`, т. е. все закладки, которые имеются в выделенном фрагменте текста. Закладки — один из самых часто используемых объектов в приложениях VBA с использованием Word. Подробнее о них будет рассказано в *разд. 10.5*.
- `Start` и `End` — свойства, которые определяют номера первого и последнего символа в выделении (по отношению к тексту документа или другим его частям, например, к сноске). Первая позиция в тексте документа — всегда 0. Если вы создаете документ из неизменяемого шаблона, можно использовать эти свойства, чтобы найти нужное место в документе для ввода текста (однако этот способ не очень рекомендуется, потому что при правке шаблона вам придется править много программного кода).
- `ExtendMode` — переключает пользователя в режим выделения текста, когда нажатие клавиш со стрелками, <Home> и <End> приводит не к перемещению указателя ввода, а к изменению выделения.
- `Find` — очень важное свойство, которое возвращает объект `Find`. Подробнее об этом объекте и о его вложенном объекте `Replace` будет рассказано в *разд. 10.6.5*.
- `Flags` — свойство, которое позволяет проверить или изменить некоторые моменты, связанные с выделением: является ли оно активным, находится ли в конце строки и т. п. Регулирует одновременно пять параметров при помощи битовой маски.
- `Font` — возвращает объект `Font`, при помощи которого можно управлять оформлением текста в выделении. Доступны все возможности, которые есть на графическом интерфейсе в меню **Формат | Шрифт**. Например, чтобы назначить выделенному тексту шрифт Arial 10 pt, можно использовать код:

```
Selection.Font.Name = "Arial"
Selection.Font.Size = 10
```

- `Information` — важнейшее свойство объекта `Selection` для целей проверок. Возвращает огромное количество информации о выделении (в какой части документа, внутри таблицы или нет, включены ли клавиши <CapsLock> и <NumLock>, включен ли режим "Замена" при вводе текста, на какой странице находится выделение и сколько страниц и т. п.).
- `IPAtEndOfLine` — возвращает `True`, если курсор ввода текста (*insertion point* — IP) находится в конце строки (в крайнем правом положении при выравнивании).
- `LanguageId` — позволяет пометить выделение, как написанное на определенном языке. Правильное определение языка позволяет избежать проблем при проверке орфографии.
- `NoProofing` — отменяет для выделения проверку орфографии и грамматики. Очень рекомендуется помечать таким образом текст с программным кодом, списками фамилий, названиями фирм, специфическими терминами и т. п.
- `Range` — создает из выделения объект `Range`.
- `StoryType` — еще одно свойство для проверок. Определяет тип текста документа, в котором находится выделение.
- `Text` — самое важное свойство объекта `Selection`. Позволяет ввести текст на месте выделения (или в том месте, где стоит указатель). Например, чтобы 100 раз напечатать текст "Привет!", можно воспользоваться кодом:

```
For i = 0 To 100
    Selection.Text = "Привет!"
    Selection.EndOf
Next
```

  Метод `EndOf()` здесь позволяет перейти в конец текущего выделения. Он нужен здесь для того, чтобы не перезаписывать один и тот же текст 100 раз, поскольку после ввода текст остается выделенным.
- `Type` — еще одно проверочное свойство, которое позволяет предупредить ошибку, если пользователь выделил что-то неположенное. Например, при обычном выделении значение этого свойства будет равно 1, а если выделены несмежные участки текста — 2.
- `Words` — позволяет вернуть коллекцию `Words`. Эта коллекция состоит из набора объектов `Range`, каждому из которых соответствует слово в выделенном тексте.

Методов у объекта `Selection` гораздо больше, чем свойств.

- `Calculate()` — позволяет посчитать математическое выражение прямо в процессе ввода текста и вернуть его результат (используя только тип данных `Single`).
- `ClearFormatting()` — очищает форматирование (и на уровне текста, и на уровне параграфа). Этот метод можно применять не только для объекта `Selection`, но и для объектов `Find` и `Replace`.
- `Collapse()` — превращает выделение в указатель вставки. Можно использовать два варианта: указатель вставки помещается на начало выделения или на конец выделения. Очень удобно, если вам требуется только вставить новый текст без удаления старого.
- `Copy()`, `CopyAsPicture()`, `Cut()`, `Paste()` и `Delete()` — эти методы можно использовать для копирования выделенного участка документа, копирования и вставки в виде изображения, вырезания, вставки и удаления соответственно.
- `EndKey()` — этот метод так называется, поскольку он очень похож по функциональности на нажатие клавиши <End>. Он позволяет (в зависимости от переданных параметров) перейти на конец строки, столбца или записи в таблице (по умолчанию на конец строки) и либо выделить до этого места, либо установить на нем указатель вставки. Чтобы перевести курсор вставки на конец текста документа, можно воспользоваться кодом:

  ```
  Selection.EndKey Unit:=wdStory, Extend:=wdMove
  ```

  Если же нужно перейти на начало элемента, используется аналогичный метод `HomeKey()`.
- `EndOf()` — по функциональности практически идентичен методу `EndKey()`. Он позволяет перейти на конец символа, слова, предложения, абзаца, секции, текста документа, таблицы и т. п. Различие между этими методами заключается в том, что `EndKey()` работает только с текущим элементом текста, а при помощи `EndOf()` можно, например, найти следующую таблицу в выделенной части документа и перейти на ее конец. Чтобы перейти на начало элемента текста, существует метод `StartOf()`.
- `Expand()` — расширяет выделение на слово, предложение, абзац и т. п., в зависимости от переданного параметра. Метод `Extend()` позволяет расширить выделение (вместо слова — предложение, вместо предложения — абзац и т. п.). Метод, обратный методу `Expand()`, — `Shrink()`.
- `GoTo()` — работает практически аналогично такому же методу объекта `Document`.

- `GotoNext()` — перейти на следующую строку, страницу, закладку и т. п. Аналогично работает метод `GotoPrevious()` (переход на предыдущий элемент).
- Назначение многочисленных методов с префиксом `Insert...` очевидно. Чаще всего используются методы `InsertBefore()` (вставить перед выделением) и `InsertAfter()` (вставить после выделения).
- Методы с префиксом `Move...` также встречаются едва ли не в любой программе, связанной с вводом текста в Word. Самые важные и удобные из этих методов:
  - `MoveLeft()`, `MoveRight()`, `MoveUp()`, `MoveDown()`, `MoveEnd()`, `MoveStart()` — эти методы позволяют переместить выделение соответственно влево, вправо, вверх, вниз, к концу существующего выделения или к его началу. Каждый из этих методов принимает дополнительные параметры, при помощи которых можно определить, на сколько символов будет перемещаться указатель, будет ли двигаться выделение, распространяясь на новую область, и т. п.;
  - `MoveStartUntil()`, `MoveStartWhile()`, `MoveEndUntil()`, `MoveStartWhile()` — отличаются тем, что курсор вставки перемещается не на определенное количество символов, а пока не будет найдено (или пока встречается) определенная последовательность символов. Также очень удобно использовать эти методы для установки курсора в нужное место в документе для ввода текста;
  - `Move()` — более гибкий метод. Он позволяет отсчитывать не только определенное количество символов, но и слов, предложений, абзацев, разделов, столбцов и строк в таблице и т. п. Позволяет обойтись минимальным количеством изменений в коде, если изменился исходный шаблон для ввода данных.
- `Next()` — позволяет перейти вперед на определенное количество символов, слов, предложений, абзацев, разделов, столбцов и строк в таблице и т. п. Переход назад осуществляет метод `Previous()`.
- `NextField()` — позволяет перейти на следующее поле в форме или проверить, не кончились ли поля (в этом случае метод вместо объекта `Field` возвратит `Nothing`). Есть также метод `PreviousField()`.
- `SelectColumn()`, `SelectRow()`, `SelectCell()` — очень удобные методы для выполнения различных операций в таблице Word.
- `SelectCurrentAlignment()`, `SelectCurrentFont()`, `SelectCurrentIndent()`, `SelectCurrentColor()` и т. п. — выделяют текст до изменения выравнивания, шрифта, отступа, цвета и т. п. Также очень удобно использовать эти

методы для форматирования или для выделения специальным образом добавленного текста.

- `SetRange()` — самый простой способ настроить выделение. Передаются номера первого и последнего символов того фрагмента текста, который нужно выделить. Нумерация начинается с 0, скрытые служебные символы также считаются. Такой же метод существует у объекта `Range`.
- `Sort()`, `SortAscending()`, `SortDescending()` — позволяют отсортировать по алфавиту, датам и т. п. абзацы или столбцы в таблице (которые входят в выделение). Этот метод поможет сэкономить вам массу времени и сил.
- `ToggleCharacterCode()` — позволяет ввести код служебного символа и тут же преобразовать его в символ Unicode. Например, чтобы ввести символ Евро, можно воспользоваться командами:

```
Selection.TypeText Text:="20ac"
Selection.ToggleCharacterCode
```

- `TypeText()` — самый простой, надежный и часто используемый метод ввода текста. Принимает единственный параметр — текст, который нужно ввести. Будет ли перезаписан текущий текст выделения, зависит от свойства `ReplaceSelection` объекта `Options` *(см. разд. 10.6.4)*.
- `WholeStory()` — выделяет текущую часть документа. Обычно используется, чтобы выделить текст документа без сносок, редакторской правки, колонтитулов и т. п.

10.5.3. Работа с объектом *Range*, его свойства и методы

Как уже говорилось, чаще всего разработчиками для определения места ввода текста и навигации по документу используется объект `Selection`. Для этих же целей можно использовать и объект `Range`. Главное отличие между этими объектами заключается в том, что объект `Selection` может определить сам пользователь (выделив текст мышью), а объект `Range` можно определить только программно, независимо от текущего положения указателя и действий пользователя.

Рекомендуется всегда использовать объект `Range` вместо объекта `Selection`. Тем самым вы защитите себя от возможных ошибок, связанных с действиями пользователя (например, если пользователь в момент, когда программно вводится текст, щелкнет мышью по какому-либо месту в документе).

Формальное определение объекта `Range` выглядит так: это программный объект, который представляет непрерывный фрагмент текста в документе. Этот

объект не зависит от объекта Selection — вы можете работать с объектом Range, не меняя текущего выделения. Он может не включать в себя ни одного символа (представлять собой указатель ввода текста).

Объектов Range в каждый момент времени может быть сколько угодно, а объектов Selection — только один.

Объект Range можно создать несколькими способами:

- первый способ — воспользоваться методом Range() объекта Document. В этом случае вам потребуется передать номера начального и конечного символов диапазона, а также текст документа, в который будут отсчитываться эти символы. Например, создать диапазон, который будет включать в себя первые 10 символов документа, можно так:

```
Dim rngDoc As Range
Set rngDoc = ActiveDocument.Range(Start:=0, End:=10)
```

- второй способ — воспользоваться свойством Range, которое предусмотрено для большого количества объектов (Bookmark, Selection, Table-Row-Cell, Paragraph и т. п.). В этом случае при помощи этого свойства мы получаем объект Range, представляющий данный объект;
- третий способ — воспользоваться большим количеством вспомогательных свойств (Characters, Words, Sentences и т. п.), которые делят текст на отрезки — объекты Range. Эти свойства возвращают коллекции объектов Range. Конечно, если вы создаете коллекцию объектов Range, представляющих каждый символ большого документа, то с точки зрения производительности такое решение может быть не самым лучшим;
- четвертый способ — переопределить существующий объект Range. Обычно для этой цели используется метод SetRange() объекта Range;
- и, наконец, пятый способ — самый удобный в реальных приложениях. Он заключается в том, что вы вначале создаете шаблон нужного вам документа (договора, приходного ордера, отчета и т. п.), в который при создании помещаете закладки в те места, где потом потребуется вставить данные. Затем программным способом для каждой закладки создается объект Range, и уже с его помощью производится ввод информации (данные о заказчике, сумма в кассовом ордере и т. п.).

Для целей отладки (чтобы убедиться, что объект Range действительно включает в себя тот фрагмент текста, который вы планировали) можно создавать на основе объекта Range объект Selection (т. е. выделять диапазон). Для этого у объекта Range предусмотрен метод Select().

Большая часть свойств и методов объекта Range совпадает с аналогичными свойствами и методами объекта Selection (который мы уже рассмотрели),

поэтому приводить здесь мы их не будем. Точно так же для объекта `Range` чаще всего используется одно-единственное свойство `Text`, которое позволяет вставить в место в документе, представленное этим объектом, нужный текст. Далее приведена информация о некоторых уникальных методах объекта `Range`.

- `InsertDatabase()` — возможно это самый простой метод вставить результат запроса к базе данных в файл Word. Генерирует таблицу на основе возвращаемого результата запроса и вставляет ее в место, определенное объектом `Range`. Умеет работать по ODBC, DDE, есть встроенные средства работы с Access. Возможности этого метода очень ограничены, поэтому рекомендуется использовать его только в самых простых случаях. Во всех остальных случаях лучше использовать объектную библиотеку ADO *(см. гл. 9)*. Пример обращения к файлу Access средствами этого метода может выглядеть так:

```
With Selection
    .Collapse Direction:=wdCollapseEnd
    .Range.InsertDatabase _
        Format:=wdTableFormatSimple2, Style:=191, _
        LinkToSource:=False, _
        Connection:="Table.Поставщики", _
        DataSource:="C:\Program Files\Microsoft
Office\OFFICE11\SAMPLES\Борей.mdb"
End With
```

- `IsEqual()` — сравнивает два объекта `Range` (или объект `Selection` и объект `Range`). Если совпадают начальная позиция в документе, конечная позиция и текст фрагмента, возвращается `True`.
- `FoneticGuide()` — позволяет вставить транскрипцию над текстом в документе.
- `Relocate()` — переставляет местами абзацы в диапазоне.
- `Select()` — как уже говорилось ранее, создает объект `Selection` на основе объекта `Range` (т. е. просто выделяет весь текст в этом объекте). Очень удобно использовать для отладочных целей.
- `SetRange()` — один из самых важных методов объекта `Range`. Позволяет изменять этот объект. Например, получить объект `Range`, в который будет входить текст от начала документа до текущей позиции курсора (или до конца выделения), можно при помощи команд:

```
Dim MyRange As Range
Set MyRange = ActiveDocument.Range(Start:=0, End:=0)
MyRange.SetRange Start:=MyRange.Start, End:=Selection.End
MyRange.Select    'для демонстрации
```

10.5.4. Объект *Bookmark*

Объект `Bookmark` — это просто закладка. На практике это самый удобный способ навигации по документам, созданных при помощи шаблонов (например, отчетов). Принципиальное отличие его от объектов `Selection` и `Range` заключается в том, что все выделения и диапазоны теряются при закрытии документа (объекты `Range` вообще существуют только во время работы создавшей их процедуры, а закладки сохраняются вместе с документом. Если документ создан на основе шаблона, то все закладки, которые были определены в этом шаблоне, будут определены и в созданном на его основе документе.

Создать закладку (с помощью меню **Вставка | Закладка**) намного проще, чем считать количество символов для объекта `Range` от начала документа, абзаца или предложения или выполнять операции `Move()` (`MoveDown()`, `MoveRight()`, `MoveNext()`) для объекта `Selection`. Кроме того, если вы будете исправлять шаблон (а делать это приходится очень часто), вам, скорее всего, не придется править код для определения места вставки (что потребуется для объектов `Selection` и `Range`).

Функциональность объекта `Bookmark` невелика. Свойств и методов у этого объекта намного меньше, чем у объектов `Selection` и `Range`. Однако обычно никто и пытается использовать объект `Bookmark` для работы с текстом. Из объекта `Bookmark` (все закладки собраны в коллекцию `Bookmarks`, принадлежащей документу) очень просто получить объект `Selection` (при помощи метода `Select()`) или объект `Range` (при помощи свойства `Range`), и дальше можно пользоваться уже свойствами и методами этих объектов, например:

```
ThisDocument.Bookmarks("Bookmark1").Select
MsgBox Selection.Text
```

Создавать объекты `Bookmark` программным способом необязательно, но если есть необходимость, то можно использовать метод `Add()` коллекции `Bookmarks`:

```
ThisDocument.Bookmarks.Add Name:="temp", Range:=Selection.Range
```

У этого метода всего лишь два параметра, которые и используются в примере.

Некоторые важные свойства объекта `Bookmark` представлены далее.

- `Empty` — если это свойство возвращает `True`, то закладка указывает на указатель вставки, а не на текст;
- `Name` — имя закладки. Очень удобно, что найти нужную закладку в коллекции `Bookmarks` можно не только при помощи индекса (номера) закладки, но и по ее имени.

- `Range` — возвращает объект `Range` на месте этой закладки.
- `Start`, `End`, `StoryType` — эти свойства аналогичны таким же у объекта `Selection`.

Методов у объекта `Bookmark` всего три.

- `Copy()` — создает закладку на основе существующей.
- `Delete()` — удаляет закладку.
- `Select()` — выделяет то, на что ссылается закладка.

10.6. Другие объекты Word

В объектной модели Word сотни объектов, и подробно о каждом из них в рамках этой книги рассказать невозможно. Скорее всего, большая часть из них вам никогда не потребуется, а другие можно будет легко найти при помощи макрорекордера. Далее приведена обзорная информация по самым важным из еще не рассмотренных объектов Word. Многие эти объекты доступны через одноименные свойства объекта `Application` и уже упоминались при рассмотрении соответствующих свойств.

10.6.1. Коллекция *AddIns* и объекты *AddIn*

Коллекция `AddIns` состоит из объектов `AddIn`, которые представляют собой шаблоны Word и приложения, встраиваемые в Word. Важная возможность этой коллекции заключается в том, что при помощи метода `Add()` можно в автоматическом режиме устанавливать шаблоны и надстройки (в графическом режиме это можно сделать через меню **Сервис | Шаблоны и надстройки**). Если вы активно используете эти средства в своих приложениях, то имеет смысл подумать над реализацией проверки наличия нужного шаблона или надстройки.

Шаблоны — это файлы с расширением dot, которые служат образцами для создания документов Word. Чаще всего они используются для того, чтобы защитить от пользователя сохраненные начальные "заготовки" отчетов, или как хранилища стилей, макросов, параметров и т. п. для сложных документов, для которых требуется стандартизация (например, в издательствах для рукописей и оригинал-макетов).

Надстройки — это откомпилированные модули DLL (для них используется расширение wll — *Word Add-In Library*). Это специальные приложения, которые изначально предназначены для встраивания в Word. Поскольку они откомпилированы и могут быть написаны на любом COM-совместимом языке, например, C++, Visual Basic или Delphi, то они работают намного быстрее,

чем родные программы VBA — макросы. Поэтому есть смысл задуматься об использовании надстроек, если вам нужно серьезное увеличение производительности.

10.6.2. Объект *AutoCorrect*

При помощи этого объекта настраиваются те параметры, которые можно на графическом экране найти в меню **Сервис | Параметры автозамены**. Эти настройки влияют на автоматическое исправление текста, вручную вводимого пользователем. Можно использовать для единообразного написания названия фирмы и прочих элементов, которые у пользователей часто получаются разными, но чаще всего этот объект используется для отключения автозамены, поскольку это может мешать пользователям. Например, чтобы очистить весь список **Заменять при вводе** на первой вкладке параметров автозамены, можно использовать код:

```
For Each Item In AutoCorrect.Entries
    Item.Delete
Next
```

10.6.3. Коллекция *Languages* и объект *Language*

Коллекция `Languages` представляет языки, которые знает данная версия Word и с которыми умеет работать. Чтобы просмотреть текущий набор языков, с которыми умеет работать Word, можно воспользоваться кодом:

```
For Each lan In Languages
    Debug.Print lan.Name
    i = i + 1
Next lan
Debug.Print i
```

10.6.4. Объект *Options*

Этот объект с огромным количеством свойств позволяет настраивать практически любые свойства самого приложения Word и текущего документа. В нем есть все, что доступно через вкладку **Сервис | Настройка** и еще множество параметров. Дочерних объектов у `Options` нет — все, что в нем есть, доступно через его свойства. Например, чтобы включить в Word отображение белых букв на синем фоне (мой привычный режим работы), можно использовать команду:

```
Options.BlueScreen = True
```

10.6.5. Объекты *Find* и *Replacement*

Объекты `Find` и `Replacement`, как понятно из их названий, предназначены для выполнения операций поиска и замены. Объект `Find` — это условия поиска, "упакованные" в программный объект. У него множество свойств (`Text`, `Style`, `Font`, `Forward`, `MathCase`, `LanguageID` и т. п.), которые позволяют определить эти условия. Чтобы запустить поиск, используется метод `Execute()` со множеством необязательных параметров, которые во многом дублируют свойства этого объекта, а передав параметр `ReplaceWith`, можно выполнить даже замену текста. Для того чтобы заменить определенное слово во всем документе или просто проверить результаты поиска, используется значение, возвращаемое методом `Execute()`. Если поиск был успешен, то возвращается `True` (и, возможно, нужно будет продолжить поиск в оставшейся части документа), а если нет — `False`.

Как будет работать объект `Find` зависит от того, из какого объекта он был создан. Если он был создан при помощи свойства `Find` объекта `Selection`, то при обнаружении нужный фрагмент выделяется. Если он был создан при помощи такого же свойства объекта `Range`, то диапазон переопределяется на найденный текст. Например, чтобы найти и выделить следующее вхождение текста `"2004"`, можно использовать код:

```
Selection.Find.Text = "2004"
Selection.Find.Execute
```

Объект `Replacement` точно так же хранит настройки замены. К этому объекту можно обратиться при помощи свойства `Replacement` объекта `Find`. Например, чтобы заменить везде до конца документа текст `"2004"` на `"2005"`, можно использовать код:

```
Selection.Find.Text = "2004"
Selection.Find.Replacement.Text = "2005"
Selection.Find.Execute Replace:=wdReplaceAll
```

Важный момент: объекты `Find` и `Replacement` по умолчанию производят поиск и замену с учетом форматирования. Поэтому если одно и то же слово будет оформлено в документе разным шрифтом, оно будет считаться разными словами. Поэтому многие программисты используют метод `ClearFormatting()`. Этот метод очищает форматирование внутри объектов `Find` и `Replacement` (на сам документ это никак не влияет) и позволяет производить поиск без учета форматирования. Например, наш код при использовании этого метода может выглядеть так:

```
Selection.Find.Text = "2004"
Selection.Find.ClearFormatting
```

```
Selection.Find.Replacement.Text = "2005"
Selection.Find.Replacement.ClearFormatting
Selection.Find.Execute Replace:=wdReplaceAll
```

10.6.6. Объекты *Font* и *ParagraphFormat*

Объекты `Font` и `ParagraphFormat` ответственны за форматирование соответственно участков текста и абзацев. Свойства объекта `Font` позволяют определить все параметры, которые доступны через пункт меню **Формат | Шрифт**, а свойства объекта `ParagraphFormat` — через **Формат | Абзац**. Объект `Font` можно получить через свойство `Font`, которое есть, в частности, у объектов `Selection`, `Range` и `Find`, а объект `ParagraphFormat` — через свойство `Format`, которое есть у объектов `Paragraph` (для одного абзаца) и коллекции `Paragraphs` (для нескольких абзацев). Свойство `Format`, которое возвращает объект `ParagraphFormat`, есть и у объекта `Find`.

Свойств у объектов `Font` и `ParagraphFormat` множество, но все они очевидны. Например, чтобы назначить выделенному тексту шрифт Arial и сделать его полужирным, можно использовать код:

```
Selection.Font.Name = "Arial"
Selection.Font.Bold = True
```

а чтобы назначить первому абзацу текста выравнивание по ширине, можно использовать команду:

```
Paragraphs(1).Format.Alignment = wdAlignParagraphJustify
```

10.6.7. Объект *PageSetup*

Если в вашем приложении используется печать, то чаще всего без использования объекта `PageSetup` не обойтись. Он позволяет программным образом настроить то, что на графическом экране настраивается через меню **Файл | Параметры страницы**. Объект `PageSetup` является вложенным в объекты `Document`, `Selection` и `Range`, и обычно обращение к нему происходит через эти объекты. Например, чтобы при печати документ был выведен в альбомной ориентации, можно воспользоваться командой:

```
ThisDocument.PageSetup.Orientation = wdOrientLandscape
```

10.6.8. Объекты *Table*, *Column*, *Row* и *Cell*

Если в вашем приложении нужно вывести какой-то стандартный документ по утвержденной форме (платежка, кассовый ордер, командировочный отчет и т. п.), то, скорее всего, для форматирования вы будете использовать табли-

цу. Таблица во многих ситуациях позволяет гарантировать правильное расположение данных в форме относительно друг друга. Многие программисты изначально используют таблицу даже для документов, которые на таблицы похожи мало (сетку таблицы всегда можно скрыть, а ячейки — слить между собой).

Создание таблицы начинается с того, что в коллекцию `Tables` (она предусмотрена для объектов `Document`, `Selection` и `Range`) добавляется новый объект `Table` (в данном примере с тремя строками и четырьмя столбцами):

```
Set Range1 = ThisDocument.Range(Start:=0, End:=0)
Dim Table1 As Table
Set Table1 = ThisDocument.Tables.Add(Range1, 3, 4)
```

Затем можно настроить свойства таблицы, например, воспользовавшись методом `AutoFormat()` (возможности у него те же, что доступны через меню **Таблица | Автоформат**):

```
Table1.AutoFormat wdTableFormatGrid5
```

Чаще всего нам нужно ввести какие-либо данные в ячейку таблицы. Ячейку таблицы в Word представляет объект `Cell`. Мы можем добраться до нужной ячейки через объекты `Columns` и `Rows`, `Selection` и `Range`, однако удобнее всего сделать так:

```
Table1.Cell(1, 1).Range.InsertAfter "10"
Table1.Cell(2, 1).Range.InsertAfter "15"
Table1.Cell(3, 1).AutoSum
```

Мы ввели в первую строку первого столбца значение 10, во вторую строку первого столбца — значение 15, а в третьей строке мы просуммировали значения по всему столбцу. Таблицы Word — это, конечно, не Excel, но при помощи метода `Formula()` объекта `Cell` в таблицу можно вставлять достаточно сложные вычисляемые значения.

Если вы используете Word для самой распространенной цели — вывод данных, полученных из другого приложения или базы данных, то, вполне возможно, заниматься программированием таблиц вам вообще будет не нужно. Достаточно будет создать и оформить таблицу в шаблоне на графическом экране и пометить места ввода данных при помощи закладок.

10.6.9. Объект *System*

При помощи объекта `System` можно получить большое количество информации о системе, в которой работает ваше приложение. В качестве альтернативы этому объекту можно подумать об использовании объектных моделей

Windows Script Host и WMI — возможностей для работы с системой в этих объектных моделях намного больше. Однако если у вас в организации используются старые операционные системы (Windows NT, Windows 98), где они могут быть не установлены, использование этого объекта очень удобно.

Далее представлены главные свойства объекта `System`.

- `CountryRegion` — возвращает текущие региональные настройки операционной системы. Если на вашем компьютере установлены русские региональные настройки, то возвращается значение 7 (несмотря на то, что его нет в документации), если установлены американские — 1.
- `FreeDisk` — возвращает объем доступного дискового пространства для пользователя на текущем диске (можно сменить текущий диск). Если документы очень большие и иногда возникают проблемы с местом на диске, можно реализовать проверку наличия свободного места, например, при выполнении операций сохранения.
- `HorizontalRezolution` и `VerticalResolution` — возможность получить информацию о текущем разрешении экрана пользователя, например, для правильного отображения больших форм.
- `LanguageDesignation` — определяет язык интерфейса операционной системы. Возвращается в виде строкового значения, например:

  ```
  Debug.Print System.LanguageDesignation
  ```

- `OperationSystem` — возвращает информацию об операционной системе. К сожалению, предусмотрено только два значения: `"Windows"` — для линейки Windows 95/98/ME и `"Windows NT"` — для линейки NT/2000/XP/2003.

У этого объекта есть всего два метода: `Connect()` (подключить сетевой диск) и `MsInfo()` (показать окно системной информации).

10.6.10. Коллекция *Tasks* и объект *Task*

Чаще всего Word запускается из Excel, Access или другого приложения, но иногда встречается и обратная необходимость — нужно открыть из Word другое приложение и переключиться в него. Самый простой способ запустить другое приложение из Word — воспользоваться стандартным объектом VBA `Shell`. Например, чтобы запустить блокнот, можно воспользоваться командой:

```
Shell "notepad.exe"
```

Есть и множество других возможностей, например, воспользоваться объектом `Application` для других приложений Word, или средствами Windows Script Host (особенно для консольных приложений), или средствами WMI, если приложение нужно запустить на другом компьютере.

После того, как приложение запущено, весь набор работающих приложений представляется в Word коллекцией `Tasks`, а каждое отдельное приложение — объектом `Task`. У коллекции `Tasks` есть два интересных метода.

- `Exists()` — проверяет, запущено ли нужное нам приложение. Например, запуск нашего блокнота с проверкой может выглядеть так:

```
If Tasks.Exists("Notepad") = False Then
    Shell "notepad.exe"
Else
    Tasks("Notepad").Activate
End If
Tasks("Notepad").WindowState = wdWindowStateMaximize
```

- `ExitWindows()` — производит операцию Log Off, т. е. завершает сеанс работы в Windows. Несохраненные документы Word при этом закроются без сохранения (и без вопросов к пользователю), а документы остальных приложений пользователю будет предложено сохранить.

У объекта `Task` интересных свойств и методов несколько больше.

- `Height`, `Width`, `Top`, `Left` — эти свойства позволяют точно настроить размер окна выбранного вами приложения.
- `Visible` — позволяет спрятать приложение.
- `WindowState` — позволяет развернуть, свернуть или восстановить окно приложения.
- `Activate()`, `Close()`, `Move()`, `Resize()` — назначение этих методов очевидно.
- `SendWindowMessage()` — самый интересный метод. Он позволяет передавать окну приложения сообщения Windows (щелчки мышью, нажатия клавиш и т. п.). Разобраться в том, какие сообщения можно посылать окнам приложений и что они означают, можно при помощи дополнительной документации из набора Microsoft Platform Software Development Kit (его можно скачать с сайта Microsoft). Например, чтобы в нашем блокноте отобразить окно **О программе**, можно воспользоваться командой:

```
Tasks("Notepad").SendWindowMessage &H111, 11, 0
```

10.6.11. Коллекция *Windows* и объект *Window*

Коллекция `Windows` и объект `Window` представляют собой окна открытых документов Word и используются в основном для настройки внешнего вида этих окон и навигации по ним. Получить доступ к окну нужного нам документа можно так:

```
Dim Window1 As Window
Set Window1 = Windows("doc2.doc")
```

или так:

```
Set Window1 = ThisDocument.ActiveWindow
```

После этого можно использовать свойства и методы объекта `Window`. Все они очень просты. Например, чтобы поменять заголовок окна, можно использовать такой код:

```
Window1.Caption = "Мое приложение"
```

Задание для самостоятельной работы 10: Программное формирование документа в Word

Ситуация:

Вам необходимо автоматизировать формирование договоров в виде документов Word. Типичный договор выглядит так, как представлено на рис. 10.2 (для простоты в этом задании вам нужно будет формировать только его начало). Изменяемые данные, которые должны подставляться программно, выделены зеленым цветом.

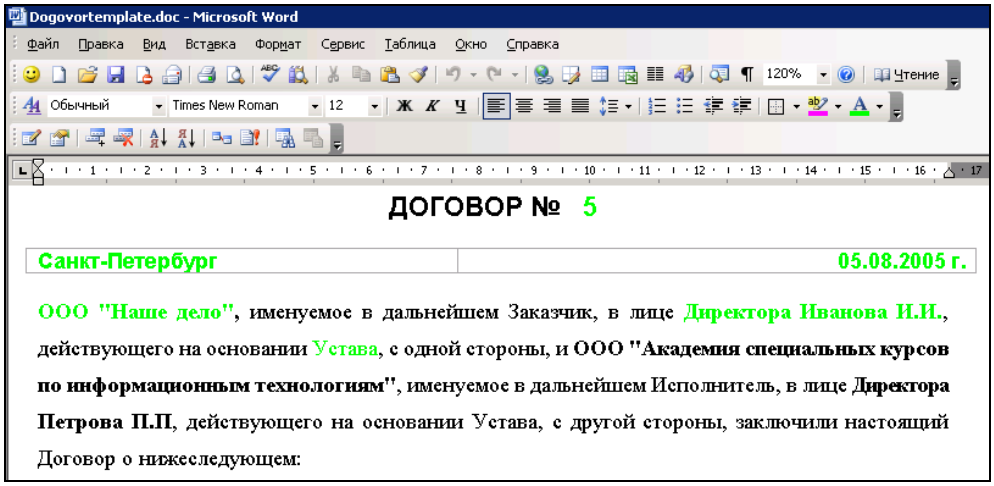


Рис. 10.2. Так должен выглядеть созданный программно договор

ЗАДАНИЕ:

1. Создайте в шаблоне Normal.dot пользовательскую форму с именем `FormDog` и заголовком **Данные договора**, аналогичную представленной на рис. 10.3.

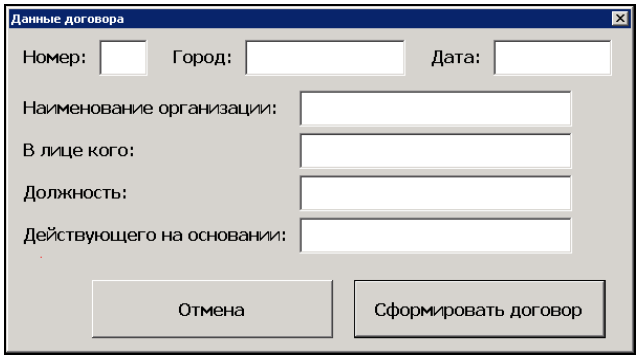


Рис. 10.3. Форма для занесения данных договора

2. Создайте макрос, по которому должна открываться эта форма, и назначьте этому макросу кнопку на панели инструментов Word.
3. Создайте и сохраните шаблон на диске с именем C:\DogovorTemplate.dot, в который будут подставляться необходимые данные, и добавьте в нужные места закладки.
4. Создайте для кнопки **Сформировать договор** на форме программный код, при помощи которого на основе шаблона и подставляемых данных из формы формировался бы новый документ с текстом договора.

Примечание

В реальном приложении данные из формы должны были бы сохраняться в базе данных и использоваться потом многократно (например, для формирования других документов). В этом задании для простоты данные для создаваемого документа берутся напрямую из формы. Работа с базой данных в похожей ситуации будет рассмотрена в задании к *гл. 12*. По этой же причине все данные в этом примере (включая дату и номер договора) текстовые.

Ответ к заданию 10

К пункту 1 задания (создание пользовательской формы):

1. Откройте окно редактора Visual Basic для Word и щелкните правой кнопкой мыши по проекту **Normal** в **Project Explorer**, а затем выберите в контекстном меню **Insert | UserForm**.
2. В дизайнере форм сконструируйте форму, аналогичную представленной на рис. 10.3 (про работу с дизайнером форм рассказывалось в *гл. 5*). Пусть в нашем примере элементы управления на форме называются так:
   - `txtNumber` — текстовое поле для ввода номера договора;
   - `txtCity` — текстовое поле для ввода города;

- txtDate — текстовое поле для ввода даты;
- txtOrg — текстовое поле для ввода наименования организации;
- txtPerson — текстовое поле для ввода представителя организации;
- txtTitle — текстовое поле для ввода его должности;
- txtLaw — текстовое поле для ввода юридического основания;
- cmdDog — кнопка для формирования договора;
- cmdCancel — кнопка **Отмена**.

3. Настройте оформление для элементов управления по вашему вкусу. Установите значение для свойства Caption для формы как "Данные договора". Для кнопки cmdDog установите True для значения свойства Default, а для кнопки cmdCancel установите True для значения свойства Cancel. Для свойства Name самой формы введите значение FormDog.

К пункту 2 задания (создание макроса и кнопки для отображения формы):

1. В стандартном модуле NewMacros проекта **Normal** создайте новую процедуру с именем FormDog(). Код ее может быть таким:

```
Public Sub FormDogShow()
    FormDog.Show
End Sub
```

   Убедитесь, что при его запуске открывается созданная вами форма.

2. В Word в меню **Сервис** выберите **Настройка**, а затем перейдите на вкладку **Команды**. В списке **Категории** выберите **Макросы**, а затем перетащите на любую панель инструментов макрос **Normal.NewMacros.FormDogShow**. Настройте для созданной кнопки подходящий формат отображения *(см. гл. 1)*. После этого закройте окно **Настройка** и убедитесь, что при нажатии на эту кнопку открывается форма.

К пункту 3 задания (создание шаблона документа Word):

1. Создайте новый документ Word, аналогичный представленному на рис. 10.4.
2. Поместите в нужные места этого документа закладки. Места вставки закладок можно посмотреть на рис. 10.2 (текст, выделенный зеленым). Пусть закладки называются так:
   - bNumber — закладка для ввода номера договора;
   - bCity — закладка для ввода города;

- `bDate` — закладка для ввода даты;
- `bOrg` — закладка для ввода наименования организации;
- `bPerson` — закладка для ввода представителя организации;
- `bTitle` — закладка для ввода его должности;
- `bLaw` — закладка для ввода юридического основания.

3. Сохраните этот файл как шаблон Microsoft Word с именем C:\DogovorTemplate.dot.

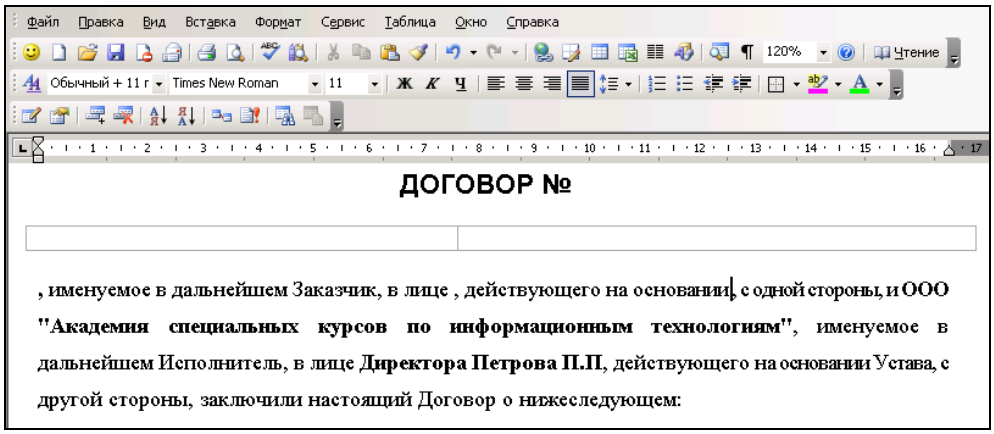


Рис. 10.4. Шаблон договора

К пункту 4 задания (создание программного кода для кнопок на форме):

1. Для события `Click` кнопки `cmdCancel` введите следующий программный код:

```
Private Sub cmdCancel_Click()
    FormDog.Hide
End Sub
```

2. Для события `Click` кнопки `cmdDog` можно использовать следующий программный код:

```
Private Sub cmdDog_Click()
    Dim oDoc As Document
    Set oDoc = Application.Documents.Add("C:\DogovorTemplate.dot")
    oDoc.Bookmarks("bNumber").Range.Text = txtNumber.Value
    oDoc.Bookmarks("bCity").Range.Text = txtCity.Value
    oDoc.Bookmarks("bDate").Range.Text = txtDate.Value
```

```
    oDoc.Bookmarks("bOrg").Range.Text = txtOrg.Value
    oDoc.Bookmarks("bPerson").Range.Text = txtPerson.Value
    oDoc.Bookmarks("bTitle").Range.Text = txtTitle.Value
    oDoc.Bookmarks("bLaw").Range.Text = txtLaw.Value
    FormDog.Hide
    oDoc.Activate
End Sub
```