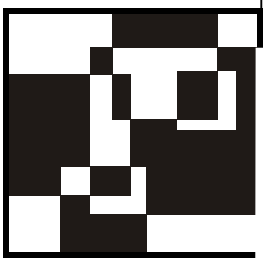
ГЛАВА 11



Программирование в Excel

11.1. Зачем программировать в Excel

Excel — это наиболее часто используемое с точки зрения программирования приложение Office. По моему опыту преподавания курсов по программированию в Office, в подавляющем большинстве случаев сотрудников предприятий интересует, как автоматизировать выполнение операций именно в Excel. Чаще всего на предприятиях встречаются следующие ситуации:

- необходимо автоматизировать загрузку данных в таблицу Excel из базы данных, а затем в автоматическом режиме произвести обработку этой таблицы (расчеты, моделирование и т. п.) и представить эту информацию в стандартном виде. На практике, конечно, намного правильнее было бы перенести выполнение расчетов (группировку, вычисление итогов по группам и т. п.) на сервер баз данных, но обычно у пользователей для этого нет ни необходимых знаний, ни прав для работы с сервером баз данных. Поэтому Excel в таких ситуациях остается незаменимым средством;
- вариант первой ситуации — приложение, работающее с базой данных, уже умеет генерировать отчеты в формате файлов Excel. Но со временем потребности в отчетах изменяются, появляется необходимость в новых отчетах или в изменении старых. Чаще всего в этом случае пользователи самостоятельно создают новые отчеты, используя данные из старых. Повторяющихся действий очень много, поэтому автоматизация таких операций бывает просто необходима;
- очень часто пользователи, не имея возможности обратиться к профессиональным программистам, самостоятельно реализуют нужные им приложения в таблицах Excel. Во множестве организаций, например, финансовое планирование или составление смет ведется просто в виде множества

файлов Excel (часто связанных между собой). Excel выполняет и роль базы данных, и роль клиентского приложения, и генератора отчетов. В таких ситуациях, конечно, опять-таки вопросы автоматизации стоят очень остро;

- формат файлов Excel удобен не только для вывода информации из базы данных, но и для загрузки введенной вручную информации в базу данных. Часто на предприятиях информация из филиалов, подразделений, от сотрудников и т. п. собирается в формате Excel. В результате со временем возникает вопрос — как автоматизировать процесс загрузки информации из Excel в базу данных;
- по моему опыту, на предприятиях часто возникает потребность в синхронизации информации между файлами Excel и базами данных (или другими файлами Excel, или файлами DBF и т. п.). Например, нужно сделать так, чтобы при занесении пользователем информации в файл Excel она сразу же добавлялась в базу данных.

Приемы, необходимые для решения подобных задач, рассматриваются в данной главе. Надеемся, что после ее изучения у вас не возникнет проблем с тем, как их решать.

С программной точки зрения Excel, в отличие от Word, чаще всего используется не для вывода и редактирования данных, а для выполнения различных расчетов и отображения их в специальных форматах (график, сводная таблица и т. п.). Если же объем данных большой (например, нужно хранить информацию по заказчикам, договорам или поставкам, то имеет смысл подумать о связке Excel плюс база данных (такая связка может быть очень удобной и производительной).

По сравнению с программным перемещением по документам Word навигацию по книгам и листам Excel производить намного удобнее, поскольку у каждой ячейки есть свой адрес (и даже два адреса — в формате A1 и в формате R1C1). Кроме того, в Excel есть возможность присваивать имена диапазонам ячеек, что также очень удобно.

Иерархия стандартных объектов в Excel немного больше. Если в Word все построено вокруг трех объектов: `Application` — `Document` — `Range`, то в Excel появляется новый элемент — лист, поэтому главная его иерархия выглядит следующим образом: `Application` — `Workbook` (книга) — `Worksheet` (лист) — `Range` (диапазон).

В Excel предусмотрена очень богатая библиотека встроенных функций (статистических, финансовых, математических и т. п.), которые можно использовать в приложениях. Часто именно наличие такой библиотеки функций оказывается решающим при выборе Excel в качестве платформы для построения приложения.

В Excel встроено несколько фактически внешних приложений, использование которых может быть очень удобным. Например, сводная таблица (объект `PivotTable`) — интегрированный в Excel OLAP-клиент приобретенной Microsoft фирмы Panorama Software, `QueryTable` — специальный объект для работы с информацией из базы данных, объект `Chart` — средство работы с диаграммами и т. п.

11.2. Объект *Application*

Как и в Word, объект `Application` в Microsoft Excel представляет все приложение Excel и находится на самом верхнем уровне объектной модели Excel. Если вам потребуется вызвать Excel из другого приложения, вам нужно будет создать объект `Excel.Application` (не забудьте при этом при помощи меню **Tools | References** добавить ссылку на библиотеку Microsoft Excel 11.0 Object Library). Создание этого объекта может выглядеть так:

```
Dim oExcel As New Excel.Application
oExcel.Workbooks.Add
oExcel.Visible = True
```

Точно так же, как и в Word, если вы работаете из уже запущенного Excel, создавать объект `Application` вам не потребуется. Он будет доступен всегда. Если вы обращаетесь к какому-либо свойству без указания вышестоящего объекта, то редактор Visual Basic в Excel будет считать, что вы обращаетесь к свойству объекта `Application`. Поэтому эти две строки кода в Excel равнозначны:

```
Application.Workbooks.Add
```

и

```
Workbooks.Add
```

Вообще объекты `Application` в большинстве приложений Office очень похожи между собой, и к ним применяются те же соображения, что и для объекта `Word.Application`. Точно так же многие разработчики считают, что удобнее и надежнее работать со скрытым окном Excel, что открывать новый экземпляр Excel удобнее, чем разыскивать уже открытый пользователем. Для того чтобы в окне редактора кода для форм появился объект `Application`, точно так же необходимо в разделе `Declarations` кода формы объявить объект `Application` с ключевым словом `WithEvents`, например, так:

```
Public WithEvents App As Excel.Application
```

В этом случае в окне редактора кода для форм у вас появится новый объект `App`, и вы сможете использовать событийные процедуры объекта `Application` (рис. 11.1).

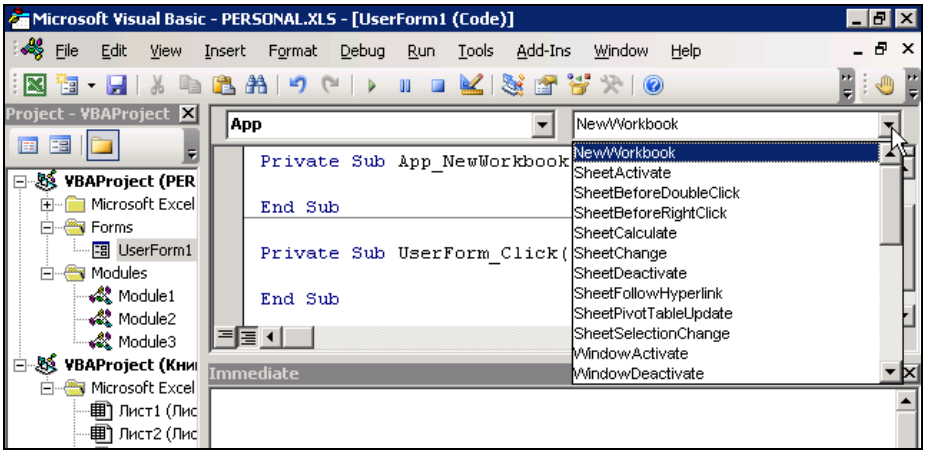


Рис. 11.1. В списке объектов появился новый объект `App` со своим набором событий

11.3. Свойства и методы объекта *Application*

Многие свойства, методы и события объекта `Excel.Application` совпадают с `Word.Application`. Однако т. к. здесь информация приводится для справки для тех пользователей, которым трудно читать по-английски, приведу наиболее часто используемые свойства и методы объекта `Application` в Excel, вне зависимости от того, встречались ли они нам в Word или нет.

Вначале о свойствах объекта `Application`.

- Свойства с префиксом `Active...` — возвращают активную ячейку (ту, на которую указывает курсор ввода данных), активную диаграмму, активный лист, активную книгу или активное окно. Все эти свойства доступны только для чтения. Собственно говоря, использовать их для создания объектов совсем не обязательно — объекты `ActiveCell`, `ActiveSheet` и т. п. создаются автоматически во время работы приложения и доступны всегда. Немного отличается свойство `ActivePrinter` — оно позволяет не только вернуть, но и установить активный принтер.
- `AddIns` — возвращает одноименную коллекцию надстроек (объектов `AddIn`). В отличие от Word, где в большинстве случаев применение надстроек предназначено для профессиональных программистов, в Excel работа с этим объектом имеет практическое значение для многих пользователей. Вместе с Excel поставляется несколько очень полезных надстроек (на графическом экране они доступны через меню **Сервис | Надстройки**), например, **Мастер подстановок**, **Пакет анализа**, **Поиск решения** и т. п. При помощи этой коллекции можно проверить, подключена ли пользова-

телем нужная надстройка (если она нужна в вашей программе) и в случае необходимости подключить ее автоматически.

- `AutoRecover` — возвращает одноименный объект, который позволяет определить параметры автосохранения Excel. Например, чтобы открытые документы Excel автоматически сохранялись каждые 5 минут, можно использовать код:

  ```
  Application.AutoRecover.Time = 5
  ```

  Время указывается в минутах, можно использовать значения в интервале от 1 до 120. На графическом экране то же самое можно сделать при помощи меню **Сервис | Параметры** на вкладке **Сохранение** окна **Параметры**.

- `Calculation` — позволяет узнать или настроить режим пересчета рабочей книги (по умолчанию установлен автоматический режим, можно также использовать ручной пересчет или полуавтоматический, когда автоматически пересчитывается все, кроме таблиц). Есть смысл отключать автоматический пересчет тогда, когда пересчет значений после каждого изменения ячейки занимает много времени и мешает вводу данных. То же самое на графическом экране можно настроить при помощи меню **Сервис | Параметры**, вкладка **Вычисления** окна **Параметры** (явно дать команду на пересчет можно клавишей <F9>).

- `CalculationState` — позволяет проверить, занимается ли Excel пересчетом данных или пересчет уже завершен.

- `Cells` — одно из самых важных свойств объекта `Application`. Оно возвращает объект `Range`, представляющий собой все ячейки в активном листе активной книги. Поскольку свойство по умолчанию (т. е. свойство, название которого можно опускать) для объекта `Range` — это свойство `Item`, то обращение к ячейкам активного листа может выглядеть так:

  ```
  Application.Cells(1, 2).Font.Bold = True
  ```

  В данном случае мы выделили полужирным ячейку на пересечении первой строки и второго столбца.

  Очень похоже действуют свойства `Columns` и `Rows`. Например, чтобы проделать подобную операцию со всем вторым столбцом, можно использовать команду вида:

  ```
  Application.Columns(2).Font.Bold = True
  ```

  а для второй строки можно воспользоваться похожей командой:

  ```
  Application.Rows(2).Font.Bold = True
  ```

  Еще раз отметим, что свойства `Cells`, `Columns` и `Rows` возвращают вовсе не наборы объектов `Cell`, `Column` и `Row`, как считают многие пользователи, а

наборы объектов Range. На использовании объекта Range построена в Excel почти вся работа с ячейками и их значениями. Далее в этой главе объекту Range будет посвящен отдельный *разд. 11.6*.

- Cursor — это свойство позволяет поменять внешний вид указателя мыши в Excel (у объекта Application в Word этого свойства почему-то нет). Обычно перед выполнением длинной расчетной операции курсору придают вид песочных часов (xlWait), а потом возвращают обратно. Автоматически по завершению работы макроса курсор не возвращается к нормальному виду, поэтому нужно предусмотреть соответствующий код.

- DataEntryMode — очень интересное свойство, которое может уберечь пользователя от множества ошибок. Оно позволяет перейти в режим ввода данных *(data entry mode)*, когда пользователю разрешается только вводить данные в разблокированные ячейки выбранного диапазона. Всего в вашем распоряжении три варианта: xlOn — включить этот режим, xlStrict — включить и сделать так, чтобы пользователь не мог из него выйти при помощи клавиши <Esc>, xlOff — отключить режим.

- DecimalSeparator и ThousandsSeparator — эти свойства позволяют не полагаться на региональные настройки на компьютере пользователя, а явно назначить символы, которые будут отделять дробную часть числа от целой и тысячи друг от друга. При использовании этих свойств рекомендуется также отключить использование системных установок при помощи свойства UseSystemSeparators:

```
Application.UseSystemSeparators = False
```

- Dialogs — возвращает одноименную коллекцию Dialog, которую можно использовать для отображения диалоговых окон Excel (их предусмотрено несколько сотен) и определять реакцию на действия в них пользователей. На этот объект очень похож объект FileDialog, представляющий окна, предназначенные только для работы с файлами (например, окно открытия файла). Работа с ними выглядит точно так же, как в Word.

- DisplayAlerts — свойство, про которое мы уже говорили в модуле про Word, но по причине его большой важности повторим еще раз. Это свойство позволяет отключить показ различных предупреждений, которые пользователю обычно в ходе работы приложения показывать не надо (например, подавить предупреждение при закрытии ненужного файла, в котором не были сохранены изменения).

- EnableEvents — позволяет на время отключить события для объекта Application, чтобы они не срабатывали (обычно перед выполнением какого-то действия — открытия файла, сохранения и т. п.).

- `ErrorCheckingOptions` — возвращает ссылку на одноименный объект, при помощи свойств которого можно настроить параметры автопроверки Excel (сообщать ли пользователю о синтаксически неверных формулах, ссылках на пустые ячейки и т. п.). По умолчанию большинство проверок включено, и к этому объекту есть смысл обращаться только тогда, когда вы хотите их отменить.
- `FileDialog` — позволяет обратиться к диалоговым окнам открытия и сохранения файлов (то же самое можно сделать при помощи более общего свойства `Dialogs`). Например, чтобы предоставить пользователю выбрать единственный файл в окне открытия и получить полный путь к нему, можно воспользоваться кодом:

  ```
  Application.FileDialog(msoFileDialogOpen).AllowMultiSelect = False
  Application.FileDialog(msoFileDialogOpen).Show
  Debug.Print Application.FileDialog(msoFileDialogOpen).SelectedItems(1)
  ```

  Похожий пример с возможностью выбора сразу нескольких файлов приведен в справке по этому свойству.
- `FileSearch` — это свойство позволяет провести поиск по указанному вами каталогу и вывести результат.
- `Interactive` — позволяет полностью заблокировать ввод в приложение Excel со стороны пользователя (как клавиатуру, так и мышь). Обычно используется, чтобы пользователь не смог помешать работе приложения, например, сбить выделение. Это свойство можно также использовать, если ввод пользователя производится из другого приложения, взаимодействующего с Excel.
- `International` и `LanguageSettings` — работают точно так же, как и в Word.
- `LibraryPath` — возвращает путь к каталогу, где лежат файлы надстроек Excel с расширением xla. По умолчанию \Office11\Library.
- `MoveAfterReturn` — позволяет включить или отключить переход на следующую ячейку после завершения ввода данных и нажатия <Enter> (по умолчанию включен), а свойство `MoveAfterReturnDirection` позволяет определить направление перехода. В некоторых ситуациях это может сильно упростить ввод данных пользователем. Например, чтобы переход происходил на ячейку справа, можно использовать команду:

  ```
  Application.MoveAfterReturnDirection = xlToRight
  ```

- `Names` — возвращает коллекцию `Names`, представляющую собой все именованные диапазоны в активной рабочей книге. При помощи метода `Add()` коллекции `Names` вы можете также сами определять в рабочей книге свои именованные диапазоны. На практике именованные диапазоны работают

примерно так же, как закладки в Word — с их помощью очень удобно определять наборы данных в сложных таблицах Excel. На графическом экране в Excel определить именованные диапазоны можно при помощи меню **Вставка | Имя**.

- `ODBCErrors` и `OLEDBErrors` — позволяют получить информацию о возникших ошибках при подключении к базам данных ODBC и OLE DB соответственно. Они возвращают одноименные коллекции, которые состоят из объектов `ODBCError` и `OLEDBError` соответственно, которые и содержат информацию об ошибке.
- `OnWindow` — это свойство больше похоже на событие. В качестве его значения указывается имя процедуры, которая должна находиться в модуле уровня книги (по умолчанию такой модуль не создается, его нужно создать вручную). Эта процедура будет вызываться всякий раз, когда пользователь переключился в окно Excel (не важно какой книги и какого листа). Вместо этого свойства можно использовать макросы со специальными именами `Auto_Activate` и `Auto_Deactivate` (если вы определили и то, и другое, первой отработает процедура, определенная при помощи свойства `OnWindow`).
- `Range` — очень важное свойство. Возвращает объект `Range`, который представляет собой диапазон ячеек и используется в Excel практически для любых операций с ячейками.
- `ReferenceStyle` — позволяет переключать режим отображения ячеек между A1 (буквы — столбцы, цифры — строки) и R1C1 (когда и строки, и столбцы обозначаются цифрами). На графическом экране это можно сделать через меню **Сервис | Параметры** на вкладке **Общие** окна **Параметры**, установив или сбросив флажок **Стиль ссылок R1C1**. На практике пользователям ближе стиль вида A1, а программистам, конечно, R1C1 (особенно в тех ситуациях, когда столбцов очень много и приходится использовать столбцы AA, AB и т. п.). Во многих ситуациях перед выполнением какой-то программной операции бывает удобно вначале перевести режим отображения в R1C1, а после окончания на радость пользователям вернуть его обратно. Можно этим и не заниматься, а использовать другие способы для отсчета определенного количества столбцов.
- `Selection` — как несложно догадаться, это свойство возвращает то, что в настоящий момент выбрал пользователь. Если он выбрал обычные ячейки в таблице, то вернется объект `Range`. Если же пользователь выбрал что-то на диаграмме, то может вернуться объект осей, легенды и т. п., в зависимости от того, что было выбрано.
- `Sheets` — это свойство мы будем подробнее разбирать в *разд. 11.5*, посвященном книгам Excel. Оно возвращает коллекцию `Sheets` — набор листов

книги и набор диаграмм, которые находятся на отдельных листах. Если используется свойство `Worksheets`, то вернется та же коллекция `Sheets`, но уже состоящая только из объектов `Worksheet` — обычных листов (без диаграмм).

- `TemplatesPath` — свойство для чтения, при помощи которого можно получить информацию о каталоге с шаблонами Excel (например, для размещения собственного шаблона или открытия шаблона из этого каталога). По умолчанию используется каталог Application Data\Microsoft\Templates в профиле пользователя.
- `ThisCell` и `ThisWorkbook` — очень удобные свойства, которые позволяют обращаться к текущей ячейке и к текущей книге, не обременяя себя созданием объектных переменных. Эти объекты создавать не нужно — они и так изначально существуют в работающем Excel.
- `Windows`, `Workbooks` и `Sheets` — возвращают, соответственно, все открытые окна, книги и листы Excel. Про объекты рабочей книги и листа мы будем говорить в *разд. 11.5*.
- `WorkSheetFunction` — позволяет использовать в программе на VBA функции Excel напрямую, без необходимости прописывать их в какую-либо ячейку.

Самые важные методы объекта `Excel.Application` перечислены далее.

- `ActivateMicrosoftApp()` — специальный метод, который предназначен для запуска и активизации (или просто активизации, если приложение уже было запущено) приложений Office (Word, Access, PowerPoint) и некоторых других (Project, FoxPro, Schedule Plus).
- `AddCustomList()` — создает новый пользовательский список. В качестве параметра принимает либо объект `Range` (элементами списка станут те значения, которые есть в диапазоне), либо стандартный объект `Array`. Удалить список можно при помощи метода `DeleteCustomList()`.
- Методы с префиксом `Calculate...` — позволяют пересчитать значения в ячейках. Простой метод `Calculate()` — это обычный пересчет значений (чаще всего нужен, если автопересчет отключен), `CalculateFull()` пересчитывает значения во всех открытых книгах, `CalculateFullRebuild()` — еще и производит перестроение формул (аналогично занесению всех формул заново). Остановить пересчет можно при помощи метода `CheckAbort()`.
- `ConvertFormula()` — преобразовывает формулу двумя способами: либо переводит адресацию ячеек в другой режим (например, вместо A1 в R1C1), либо меняет абсолютные ссылки на относительные и наоборот. В качестве параметра принимает строковую переменную с текстом формулы (должна начинаться с символа '=') и флаги конвертации.

- `DoubleClick()` — этот метод эквивалентен двойному щелчку мышью на активной ячейке, т. е. переход в режим ввода данных в эту ячейку.
- `Evaluate()` — очень полезный и часто используемый метод. Позволяет по имени найти объект книги Excel и преобразовать его в объект или значения для дальнейшего использования. В качестве имен этот метод может принимать:
  - имена ячеек в стиле A1 (возвращается объект `Cell`);
  - имена диапазонов (возвращается объект `Range`);
  - имена, определенные в макросе (чаще всего названия переменных);
  - ссылки на внешние книги, например:

    ```
    Evaluate("[Book1.xls]Sheet1!A5"))
    ```

  Этот метод можно вызвать и неявно, просто заключив имя объекта в квадратные скобки. Например, такие строки будут абсолютно одинаковыми по значению:

  ```
  [a1].Value = 25
  Evaluate("A1").Value = 25
  ```

  Поскольку синтаксис с квадратными скобками короче, чаще всего используется именно он.

  Таким образом, метод `Evaluate()` — это самый простой и естественный метод обратиться к ячейке или диапазону в своей или чужой книге Excel.
- `GetOpenFilename()` — это упрощенный способ работы с окном открытия файлов. Чтобы открыть диалоговое окно и получить информацию о том, что выбрал пользователь (в виде строковой переменной с информацией о имени файла с полным путем), можно использовать такой код:

  ```
  Filename = Application.GetOpenFilename()
  If Filename <> False Then
      Debug.Print Filename
  End If
  ```

- `GetSaveAsFilename()` — такой же по функциональности способ, который работает с окном **Сохранить как**.
- `GoTo()` — важный и часто используемый метод. Принимает в качестве параметра объект `Range`, строку со ссылкой на ячейку (в формате R1C1) или имя процедуры VBA, выделяет и активизирует диапазон или ячейку или запускает на выполнение процедуру. В отличие от `Select()`, этот метод еще и автоматически активизирует лист, на котором расположены диапазон или ячейка.

- `Help()` — позволяет открыть и показать информацию из файла справки (в том числе пользовательского). В качестве параметра принимает имя файла справки и метку темы в нем.
- `Intersect()` — возвращает диапазон, который представляет собой область пересечения двух или более диапазонов. Если передаваемые в качестве параметров диапазоны не пересекаются, возвращается `Nothing`.
- `OnKey()` — этот метод позволяет "посадить" процедуру VBA на определенную клавиатурную комбинацию. Например, чтобы назначить процедуру `Msg()` из модуля `Лист1` клавиатурной комбинации <Alt>+<M>, можно воспользоваться командой:

```
Application.OnKey "%{m}", "Лист1.Msg"
```

- `OnRepeat()` — позволяет назначить процедуру, которая будет выполняться при использовании команды **Повторить** в меню **Правка**. Назначение процедуры команде **Отменить** в том же меню производится при помощи метода `OnUndo()`.
- `RegisterXLL()` — подключает файл надстройки Excel с расширением xll и регистрирует его функции и процедуры. Этот метод используется при установке собственного приложения.
- `Repeat()` — позволяет просто повторить последнее действие, которое было выполнено пользователем (не программным способом). Аналогично команде **Повторить** в меню **Правка**. Отменить действие пользователя можно при помощи метода `Undo()`.
- `Run()` — позволяет выполнить процедуру или функцию VBA, макрос Excel или процедуру или функцию в XLL-модуле (и передать до 30 параметров).
- `SendKeys()` — позволяет эмулировать нажатия клавиш и передать их в активное окно приложения. Обычно используется, если нужно что-то продемонстрировать пользователю или что-то сделать через меню (например, не удалось найти, как это можно выполнить с помощью кода).
- `Union()` — позволяет объединить два или более диапазонов.
- `Volatile()` — этот хитрый метод должен вызываться только из пользовательской функции, которая вычисляет значение ячейки. Если он вызван и ему передано значение `True`, то данная ячейка становится чувствительной *(volatile)* и будет пересчитываться при любом изменении значений в листе, даже том, которое ее не касается.
- `Wait()` — этот метод позволяет приостановить работу Excel на указанное вами время, сняв нагрузку с процессора. Используется для демонстраций, чтобы пользователь успел увидеть, что происходит, для ожидания завершения выполнения какой-либо внешней операции и т. п. При этом ввод

пользователя блокируется, а указатель мыши приобретает вид песочных часов. Однако некоторые фоновые операции Excel, такие как печать и пересчет значений, будут продолжать выполняться. Например, чтобы взять паузу на пять секунд, можно воспользоваться кодом:

```
If Application.Wait(Now + TimeValue("0:00:5")) Then
    MsgBox "Пять секунд прошло"
End If
```

- `Quit()` и `OnTime()` — делают то же самое, что и в Word.

11.4. Коллекция *Workbooks* и объект *Workbook*, их свойства и методы

Следующий по иерархии после `Application` объект в объектной модели Excel — это объект `Workbook`, который представляет собой книгу Excel. Можно сказать, что объект `Workbook` занимает в Excel примерно то же место, что и объект `Document` в Word — он необходим для получения ссылки на нужную нам книгу в наборе открытых книг Excel, для настройки общих свойств и выполнения общих действий со всеми листами книги. Получить этот объект можно очень просто:

- первый способ — воспользоваться коллекцией `Workbooks`, которая доступна через свойство `Workbooks` объекта `Application`. Впрочем, применять это свойство совершенно не обязательно — коллекция `Workbooks` в Excel и так постоянно доступна. Найти нужную книгу в этой коллекции можно по ее имени или номеру в коллекции, например:

  ```
  Debug.Print Workbooks("Смета.xls").FullName
  ```

- второй способ — использовать свойство `Application.ActiveWorkbook`. При помощи этого свойства мы обращаемся к активной в настоящей момент книге:

  ```
  Debug.Print ActiveWorkbook.Name
  ```

- третий способ — использовать свойство `Application.ThisWorkbook`. При этом мы обращаемся к книге, которой принадлежит данный программный модуль:

  ```
  Debug.Print ThisWorkbook.Name
  ```

На практике чаще всего нам нужно либо создать в Excel новую книгу, либо открыть существующую книгу (или другой файл в формате, который понимает Excel, например, DBF). Для этой цели используются методы `Add()` и `Open()` соответственно. Например, создать новую книгу в Excel можно так:

```
Dim oWbk As Workbook
Set oWbk = Workbooks.Add()
```

Единственный необязательный параметр, который принимает этот метод, — имя шаблона, на основе которого создается новая рабочая книга.

Открытие существующей книги выглядит так:

```
Dim oWbk As Workbook
Set oWbk = WorkBooks.Open("C:\mybook1.xls")
```

Кроме стандартных, в коллекции `Workbooks` предусмотрено также три специальных метода.

- `OpenDatabase()` — открывает базу данных, выполняет запрос к ней (или напрямую открывает таблицу или представление), а результаты запроса помещает как импортированные внешние данные в новую автоматически созданную рабочую книгу Excel;
- `OpenText()` — почти то же самое, но в качестве источника здесь выступает текстовый файл. Дополнительные параметры позволяют определить его формат.
- `OpenXML()` — в качестве источника данных будет выступать файл в формате XML.

Как и метод `InsertDatabase()` в Word, эти методы следует использовать только в самых простых случаях. Рекомендуется по возможности применять более мощные и стандартные средства объектной модели ADO.

Теперь о самых важных свойствах объекта `Workbook` — самой рабочей книги.

- `Name`, `CodeName`, `FullName` — разные имена этой книги. Самое простое имя — `Name`, которое совпадает с именем файла книги. `FullName` — это имя файла книги вместе с полным путем к нему в операционной системе. `CodeName` — как эта книга называется в коде. `CodeName` можно посмотреть в окне **Project Explorer** или, если открыть свойства книги в окне **Properties**, кодовое имя книги будет представлено в строке **(Name)**. Все три свойства доступны только для чтения, менять их можно другими способами (например, сохраняя файл под другим именем или изменив свойство в окне **Properties**).

  Определенное отношение к именам имеет также свойство `Path`, которое возвращает полный путь в файловой системе к книге Excel.

- `Charts`, `Sheets`, `ActiveChart`, `ActiveSheet`, `CustomViews`, `BuiltinDocumentProperties` и `CustomDocumentProperties`, `Windows`, `WebOptions` — возвращают одноименные коллекции соответствующих объектов. Некоторые из них будут рассматриваться в следующих разделах.

- `ConflictResolution` — определяет, как будут разрешаться конфликты изменения данных, если книга открыта сразу несколькими пользователями *(shared workbook)*. Есть возможность сделать так, чтобы локальный пользователь автоматически выигрывал, автоматически проигрывал или возникало диалоговое окно с возможностью разобраться в конфликте вручную. Существует большое количество свойств, которые позволяют настроить параметры совместной работы с книгой, но по причине того, что такая работа не рекомендуется (данные для совместного доступа необходимо переносить в базу данных), рассматриваться они здесь не будут, за исключением:
  - запрещать или разрешать общий доступ к рабочей книге можно при помощи методов `SaveAs()` или `ExclusiveAccess()`;
  - по умолчанию возможность совместного редактирования для книги отключена (проверить можно при помощи свойства `MultiUserEditing`);
  - получить список всех пользователей (а также информацию, когда они открыли файл и в каком режиме) можно при помощи свойства `UserStatus`.
- `FileFormat` — возвращает формат книги (доступен напрямую только для чтения, можно изменять при сохранении книги). Форматов очень много: разные версии Excel, DBF, Lotus 1-2-3, форматы TXT, CSV, XML — всего несколько десятков.
- `Names` — возвращает коллекцию всех именованных диапазонов в данной рабочей книге. Получить информацию о таких диапазонах можно, например, так:

```
For Each Item In ThisWorkbook.Names
    Debug.Print Item.Name
Next
```

  Это свойство удобно использовать для предварительных проверок для устранения потенциальных ошибок времени выполнения.

Методов у объекта `Workbook` также очень много, однако самые часто используемые из них — это `Activate()`, `Close()`, `Save()`, `SaveAs()`, `PrintOut()`, `Protect()` и `Unprotect()`, которые очевидны и действуют аналогично одноименным методам объекта `Document` в Word.

11.5. Коллекция *Sheets* и объект *Worksheet*, их свойства и методы

В Word на уровне ниже объекта `Application` и `Document` начинались уже объекты непосредственно для работы с текстом: `Selection`, `Range` и т. п. В Excel

между объектом рабочей книги и ячейками есть еще один промежуточный объект — `Worksheet` (лист). Объекты `Worksheet` в книге объединены в коллекцию `Sheets`.

Чаще всего для ввода данных в Excel (напрямую или из базы данных) нам потребуется, в первую очередь, определиться с листом, на который будет выполняться ввод данных — либо просто выбрать его, либо вначале создать, а потом выбрать.

Процесс создания нового листа выглядит очень просто:

```
Dim oExcel As New Excel.Application     'Запускаем Excel
oExcel.Visible = True                 'Делаем его видимым
Dim oWbk As Excel.Workbook
Set oWbk = oExcel.Workbooks.Add()    'Создаем новую книгу
Dim oSheet As Excel.Worksheet
Set oSheet = oWbk.Worksheets.Add()  'Создаем новый лист
oSheet.Name = "Новый лист"           'Присваиваем ему имя "Новый лист"
```

Метод `Add()` для коллекции `Worksheets` может принимать несколько необязательных параметров, главная задача которых — определить, между какими существующими листами книги будет вставлен новый лист. Если ничего не указывать, то новый лист будет помещен самым первым.

Часто встречается и другая задача — найти нужный лист среди листов книги, например, если мы открыли существующую книгу. Сделать это очень просто, поскольку коллекция `Worksheets` умеет работать с именами листов. Далее приведен пример, в котором мы запускаем Excel и создаем новую книгу, но при этом находим лист с именем `"Лист1"` и переименовываем его в `"Новый лист"`:

```
Dim oExcel As New Excel.Application         'Запускаем Excel
oExcel.Visible = True                       'Делаем его видимым
Dim oWbk As Excel.Workbook
Set oWbk = oExcel.Workbooks.Add()           'Создаем новую книгу
Dim oSheet As Excel.Worksheet
Set oSheet = oWbk.Worksheets.Item("Лист1")  'Находим Лист1
oSheet.Name = "Новый лист"      'Присваиваем ему имя "Новый лист"
```

Обратите внимание, что в английской версии Excel этот код не пройдет, поскольку листы там по умолчанию называются `"Sheet1"`, `"Sheet2"` и т. п. Если вы используете в коде имена листов, заданные по умолчанию, и при этом вашей программе придется работать на компьютерах с разноязычными версиями Excel, обязательно предусмотрите дополнительные проверки или просто используйте номера листов вместо их имен.

У коллекции `Sheets` есть привычные нам свойства и методы коллекций VBA (`Count`, `Item`, `Add()`, `Delete()`). Другие свойства и методы удобнее применять для объекта `Worksheet` (`Visible()`, `Copy()`, `Move()`, `PrintOut()`, `PrintPreview()`, `Select()`), поскольку все равно нам придется указывать конкретный лист. Однако для этой коллекции предусмотрен и один специфический метод `FillAcrossSheets()` — скопировать объект диапазона `Range` (полностью, только содержимое или только оформление) во все листы данной книги.

У объекта `Worksheet` есть множество важных свойств и методов.

- `Cells` — одно из наиболее часто используемых свойств. Работает точно так же, как и рассмотренное ранее одноименное свойство объекта `Application`, за исключением того, что вам не придется ограничиваться только активным листом. Аналогично работают свойства `Columns` и `Rows`.
- `EnableCalculation` — позволяет отключить автоматический пересчет значений ячеек на листе.
- `EnableSelection` — позволяет запретить выделять что-либо на листе, снять запрет или разрешить выделять только незаблокированные ячейки.
- `Next` — получает ссылку на следующий лист в книге, в свойство `Previous` — на предыдущий лист.
- `PageSetup` — как и в Word, позволяет получить объект `PageSetup`, при помощи которого можно настроить те же параметры, что и через меню **Файл | Параметры страницы**.
- `Protection` — позволяет получить объект `Protection`, при помощи которого можно запретить пользователю вносить изменения в лист Excel. Для настройки параметров защиты предназначены также и другие свойства, названия которых начинаются с префикса `Protection...`
- `QueryTables` — исключительно важное свойство. Оно возвращает коллекцию `QueryTables` — набор объектов `QueryTable`, которые, в свою очередь, представляют данные, полученные из внешних источников (как правило, из баз данных).
- `Range` — самое важное свойство объекта `Worksheet`. Возвращает объект `Range` (диапазон ячеек), который в объектной модели Excel занимает примерно такое же место, что и одноименный объект в объектной модели Word. Этот объект будет рассматриваться далее в *разд. 11.6*.
- `Type` — определяет тип данного листа. Обычно используются два типа: `xlWorksheet` (обычный лист) и `xlChart` (диаграмма).
- `UsedRange` — возвращает объект `Range`, представляющий собой прямоугольную область, включающую все непустые ячейки листа. Удобно использовать для копирования или форматирования.

- `Visible` — позволяет спрятать лист от пользователя (например, если он используется для служебных целей).

Некоторые важные методы объекта `Worksheet` представлены далее.

- `Activate()`, `Calculate()`, `Copy()`, `Paste()`, `Delete()`, `Move()`, `Evaluate()`, `Select()`, `SaveAs()`, `PrintOut()`, `PrintPreview()`, `Protect()`, `Unprotect()` — эти методы нам уже знакомы. Отличие заключается только в том, что теперь эти методы могут применяться для выбранного вами листа.
- `PivotTables()` — возвращает коллекцию очень интересных объектов `PivotTable` (сводная таблица), которые будут рассматриваться в *разд. 11.8*.
- `Scenarios()` — возвращает коллекцию `Scenarios`, состоящую из объектов `Scenario` (сценарии). Сценарии — это именованные наборы вводных данных, которые можно использовать для проверки различных вариантов (разные суммы продаж, уровни налогов, расходов и т. п.).
- `SetBackgroundPicture()` — позволяет назначить листу фоновое изображение (естественно, желательно, чтобы оно было полупрозрачным, как "водяной знак", иначе на его фоне будет трудно читать текст в ячейках).
- `ShowAllData()` — показывает все скрытые и отфильтрованные данные на листе.

Самое важное событие объекта `Worksheet` — это, конечно, `Change`. Существует множество практических задач, когда изменение пользователем значения в ячейке должно приводить к изменению значения в ячейке другого листа или рабочей книги Excel, или даже в базе данных. Другая ситуация, в которой используется это событие, — сложная проверка вводимого пользователем значения (например, когда это значение сверяется со значением в базе данных). Эта событийная процедура работает со специальным параметром `Target`, т. е. с объектом `Range`, представляющим изменившуюся ячейку. При помощи свойств и методов объекта `Range` вы можете получить информацию об изменившемся значении, столбце и строке, в котором произошло изменение, и т. п.

У объекта `Worksheet` есть еще два очень удобных события (их сильно не хватает объекту `Document` в Word) — это `BeforeRightClick()` и `BeforeDoubleClick()`. Как понятно из названий, первое событие позволяет перехватывать щелчок правой кнопкой мыши по любому месту в листе, а второе событие — двойной щелчок мышью. При помощи этих событий вы можете назначить свою реакцию (открытие контекстных меню, выдачу предупреждающих сообщений, переход в другой режим работы и т. п.) на действия пользователя.

11.6. Объект *Range*, его свойства и методы

Пожалуй, наиболее часто используемый объект в иерархии объектной модели Excel — это объект `Range`. Он может представлять одну ячейку, несколько ячеек (в том числе несмежные ячейки или наборы несмежных ячеек) или целый лист. Если в Word вы могли для ввода данных использовать как объект `Range`, так и объект `Selection`, то в Excel все сводится к объекту `Range`:

- если вам нужно ввести данные в ячейку или отформатировать ее, то вы должны получить объект `Range`, представляющий эту ячейку;
- если вы хотите сделать что-то с выделенными вами ячейками, вам необходимо получить объект `Range`, представляющий выделение;
- если вам нужно просто что-то сделать с группой ячеек, первое ваше действие — это получить объект `Range`, представляющий эту группу ячеек.

В базе знаний Microsoft (**www.microsoft.com/support**) есть статья под номером 291308, в которой описываются 22 способа получения объекта `Range` в Excel. Вряд ли вы будете пользоваться всеми эти способами. Мы рассмотрим только самые распространенные.

- Самый простой и очевидный способ — воспользоваться свойством `Range`. Это свойство предусмотрено для объектов `Application`, `Worksheet` и для самого объекта `Range` (если вы решили создать новый диапазон на основе уже существующего). Например, получить ссылку на объект `Range`, представляющий ячейку A1, можно так:

```
Dim oRange As Range
Set oRange = Worksheets("Лист1").Range("A1")
```

  А ссылка на диапазон ячеек с A1 по D10 создается так:

```
Dim oRange As Range
Set oRange = Worksheets("Лист1").Range("A1:D10")
```

  С применением свойства `Range` самого объекта `Range` нужно быть очень осторожным. Дело в том, что Excel создает на основе объекта `Range` виртуальный лист со своей собственной нумерацией. Поэтому такой код:

```
Set oRange1 = Worksheets("Лист1").Range("C1")
Set oRange2 = oRange1.Range("B1")
oRange2.Value = 20
```

  пропишет значение 20 не в ячейку B1, как можно было понять из кода, а в ячейку D1 (т. е. в ячейку B1 по отношению к виртуальному листу, начинающемуся с C1).

- Второй способ — воспользоваться свойством `Cells` объекта `Worksheet`. Возможностей у этого свойства меньше — возвращается диапазон, состоящий только из одной ячейки. Зато мы можем использовать более удобный синтаксис (с точки зрения передачи переменных, перехода в любую сторону на любое количество ячеек и т. п.). Например, для получения ссылки на ячейку D1 можно использовать код вида:

  ```
  Dim oRange As Range
  Set oRange = Worksheets("Лист1").Cells(1, 4)
  ```

  Чтобы получить диапазон, состоящий из нескольких ячеек, удобно применять свойства `Range` и `Cells` вместе:

  ```
  Dim oRange
  Set oRange = Range(Cells(1, 1), Cells(5, 3))
  ```

- Третий способ — воспользоваться многочисленными свойствами объекта `Range`, которые позволяют изменить текущий диапазон или создать на основе его новый. Эти свойства будут рассмотрены далее в этом разделе.

Обычно после того, как нужная ячейка найдена, в нее нужно что-то записать. Для этой цели используется свойство `Value`, например:

```
oRange.Value = "Мое значение"
```

Поскольку объект `Range` с функциональной точки зрения очень важен, то свойств и методов у него очень много (и для комфортной работы в Excel их нужно знать). Далее представлены некоторые самые часто употребимые свойства.

- `Address` — позволяет вернуть адрес текущего диапазона. Например, для предыдущего примера вернется $A$1:$C$5. Если в диапазоне только одна ячейка, то вернется значение вида $A$1. Этому свойству можно передать много разных параметров для определения стиля ссылки, абсолютного или относительного адреса для столбцов и строк, по отношению к чему этот адрес будет относительным и т. п. Свойство доступно только для чтения. `AddressLocal` — то же самое, но с поправкой на особенности локализованных версий Excel.

  На практике встречается множество ситуаций, когда адрес ячейки нужно разобрать на части и вернуть из него имя столбца или номер строки. Это очень просто сделать при помощи строковых функций и знака доллара.

  Например, имя столбца для объекта `oRange`, представляющего одну ячейку, можно вернуть так:

  ```
  sColumnName = Mid(oRange.Address,2,(InStr(2,oRange.Address,"$") – 2))
  ```

А номер строки — так:

```
sRowNumber = Mid(oRange.Address,(InStr(2,oRange.Address,"$") + 1))
```

На первый взгляд это кажется сложным, но на самом деле все очень просто — для имени столбца мы просто берем все, что у нас находится между первым знаком доллара (он всегда идет первым символом) и вторым, а для номера строки — все, что находится после второго знака доллара. Найти второй знак доллара можно при помощи встроенной функции `InStr()`, а взять нужное количество символов, начиная с определенной позиции, проще всего при помощи встроенной функции `Mid()`.

- `AllowEdit` — это свойство, доступное только для чтения, позволяет определить, сможет ли пользователь вносить исправления в данную ячейку (набор ячеек) на защищенном листе. Используется для проверок.
- `Areas` — очень важное свойство. Дело в том, что объект `Range` может состоять из несмежных наборов ячеек. Многие методы применительно к таким диапазонам ведут себя совершенно непредсказуемо или возвращают ошибки. Свойство `Areas` позволяет разбить подобные нестандартные диапазоны на набор стандартных. Созданные таким образом объекты `Range` будут помещены в коллекцию `Areas`. Это свойство можно использовать и для проверки "нестандартности" диапазона:

```
If Selection.Areas.Count > 1 Then
    Debug.Print "Диапазон с несмежными областями"
End If
```

- `Borders` — возвращает ссылку на коллекцию `Borders`, при помощи которой можно управлять рамками для диапазона.
- `Cells` — это свойство, как мы уже видели, есть и у объекта `Worksheet`. Для объекта `Range` оно работает точно так же, за исключением того, что опять-таки используется своя собственная виртуальная адресация на основе диапазона:

```
Dim oRange, oRange2 As Range
Set oRange = Range(Cells(2, 2), Cells(5, 3))
Set oRange2 = oRange.Cells(1, 1)  'Вместо A1 получаем ссылку на B2
Debug.Print oRange2.Address       'Проверка
```

Точно такие же особенности есть у свойств `Row` и `Rows`, `Column` и `Columns`.

- `Characters` — это простое с виду свойство позволяет решать непростую задачу: как изменить фрагменты текста или их формат в ячейке, не затрагивая остальные данные. Например, чтобы ввести текст в ячейку A1 и изменить цвет первой буквы, можно воспользоваться кодом:

```
Dim oRange As Range
Set oRange = Range("A1")
oRange.Value = "Мой текст"
oRange.Characters(1, 1).Font.Color = vbRed
```

Если же вам просто нужно изменить значение, то лучше воспользоваться свойством `Value` — как в третьей строке примера.

- `Count` — возвращает количество ячеек в диапазоне. Может использоваться для проверок.
- `CurrentRegion` — очень удобное свойство, которое может пригодиться, например, при копировании или экспорте данных, полученных из внешнего источника (когда нам изначально неизвестно, сколько будет данных). Оно возвращает объект `Range`, представляющий диапазон, окруженный пустыми ячейками (т. е. непустую область, в которую входит исходный диапазон или ячейка). Например, чтобы выделить всю непустую область вокруг активной ячейки, можно воспользоваться кодом:

```
Worksheets("Лист1").Activate
ActiveCell.CurrentRegion.Select
```

Затем, к примеру, можно программным образом создать новый столбец справа от последнего имеющегося, или под последней заполненной строкой создать новую строку с итогами.

- `Dependents` — позволяет получить объект `Range` (скорее всего, включающий несмежные области), представляющий ячейки, которые зависят от ячеек исходного диапазона. Работает только для текущего листа — ссылки во внешних листах этим свойством не отслеживаются. Например, чтобы выделить все ячейки, зависимые от активной, можно использовать код:

```
Worksheets("Лист1").Activate
ActiveCell.Dependents.Select
```

Чтобы просмотреть обратную зависимость, можно использовать свойство `Precedents`. Чтобы просмотреть только первый уровень зависимостей, можно использовать свойства `DirectDependents` и `DirectPrecedents`.

- `End` — еще одно часто используемое свойство. Оно позволяет получить объект `Range`, представляющий последнюю ячейку исходного диапазона. В каком направлении будет возвращаться последняя ячейка, можно определить при помощи передаваемого параметра.
- `Errors` — свойство, которое через коллекцию `Errors` позволяет получить доступ к объектам `Error`, представляющим обнаруженные ошибки в диапазоне.

- `Font` — как и в Word, это свойство позволяет получить доступ к объекту `Font`, при помощи которого можно настроить особенности оформления текста в ячейке (цвет, шрифт, размер букв и т. п.).
- `FormatConditions` — позволяет создать собственный объект, представляющий вариант оформления ячеек, который затем можно применять к разным ячейкам и диапазонам.
- `Formula` — одно из самых важных свойств объекта `Range`. Доступно и для чтения, и для записи. Возвращает текст формулы, прописанной в ячейку (а не вычисленное значение) или позволяет записать формулу в ячейку. Если применить это свойство для диапазона из нескольких ячеек, то формула будет прописана по всей ячейке диапазона. Пример использования этого свойства может выглядеть так:

```
Worksheets("Лист1").Range("A3").Formula = "=$A$1+$A$2"
```

- `FormulaHidden` — позволяет спрятать формулы от пользователя в данном диапазоне. Работает только на защищенных листах.
- `HasFormula` — проверяет диапазон на наличие вычисляемых значений (формул).
- `Hidden` — позволяет спрятать диапазон. Будет работать только в случае, если диапазон включает в себя хотя бы одну строку или столбец целиком, в противном случае вернется ошибка.
- `Interior` — еще одно свойство, связанное с форматированием. Позволяет выделить цветом ячейки диапазона.
- `Item` — возвращает еще один объект `Range`, который определяется путем смещения исходного диапазона.
- `Locked` — позволяет заблокировать ячейки диапазона при защите листа.
- `Name` — позволяет получить ссылку на специальный объект именованного диапазона `Name`. На графическом экране с его возможностями можно познакомиться при помощи меню **Вставка | Имя**. Он позволяет обращаться к диапазонам и формулам по именам и несколько напоминает по функциональности объект `Bookmark` в Word.
- `Next` — позволяет перейти на следующую ячейку. Если лист не защищен, то следующей ячейкой будет считаться ячейка справа, если лист защищен — то следующая незаблокированная ячейка.
- `NumberFormat` — устанавливает один из предопределенных форматов для чисел. Соответствует возможностям вкладки **Число** в меню **Формат | Ячейки** на графическом экране.
- `Offset` — это свойство позволяет получить новый объект `Range` с определенным смещением относительно исходного диапазона. Например, чтобы

получить ячейку со смещением на три ячейки вверх и три ячейки влево, можно использовать код:

```
Worksheets("Лист1").Activate
ActiveCell.Offset(rowOffset:=-3, columnOffset:=-3).Activate
```

- `Orientation` — позволяет сориентировать текст в ячейках. Определяет угол наклона в градусах. Например, чтобы расположить текст по диагонали, можно использовать код:

```
oRange.Orientation = -45
```

- `PageBreak` — это свойство обычно используется для программной вставки разрывов страницы. Его применение может выглядеть так:

```
Worksheets("Лист1").Rows(50).PageBreak = xlPageBreakManual
```

- свойства с префиксом `Pivot...` — относятся к работе с объектом `PivotTable` (сводная таблица). Особенности работы с ним будут рассмотрены в *разд. 11.8*.

- `QueryTable` — это важное свойство позволяет получить ссылку на объект `QueryTable` — полученные с внешнего источника данные, которые находятся в данном диапазоне. Подробнее про объект `QueryTable` будет рассказано в *разд. 11.7*.

- `Range` — это свойство, как уже говорилось ранее, позволяет создать новый диапазон на основе уже существующего. Необходимо помнить про особенности нумерации ячеек в этом случае.

- `Resize` — позволяет изменить текущий диапазон. Например, увеличение его на один столбец вниз и на одну строку вправо можно выполнить так:

```
oRange.Resize(oRange.Rows.Count + 1, oRange.Columns.Count + 1).Select
```

- `ShrinkToFit` — это свойство позволяет автоматически настроить размер текста в диапазоне таким образом, чтобы текст помещался в ширину столбца.

- `Style` — позволяет вернуть объект `Style`, представляющий стиль для указанного диапазона. На графическом экране то, что выполняет объект `Style`, можно сделать через меню **Формат | Стиль**.

- `Text` — получает значение первой ячейки диапазона в виде значения типа `String`. Для объекта `Range` это свойство доступно только для чтения.

- `Validation` — это свойство позволяет вернуть объект `Validation`, при помощи которого можно настроить проверку вводимых в диапазон данных.

- `Value` — наиболее часто используемое свойство объекта `Range`. Позволяет получить или назначить значение (числовое, текстовое или какое-либо

другое) ячейкам диапазона. Точно также используется свойство `Value2`, но с единственным отличием — оно не поддерживает типы данных `Currency` и `Date`.

- `WrapText` — позволяет включить или отключить перевод текста на следующую строку в ячейках диапазона.

Информация о методах объекта `Range` представлена далее.

- `Activate()` — выделяет текущий диапазон и устанавливает курсор ввода на его первую ячейку.
- `AddComment()` — добавляет комментарий к ячейке. Ячейка будет помечена красным уголком, а текст комментария будет показываться в виде всплывающей подсказки. Этот метод можно вызвать только для диапазона, состоящего из одной ячейки. То же самое на графическом экране можно сделать при помощи меню **Вставка | Примечание**.
- `AutoFill()` — позволяет использовать автозаполнение для диапазона (например, если первые две ячейки будут заполнены как 1 и 2, то дальше в автоматическом режиме будет продолжено: 3, 4, 5 и т. п.).
- `AutoFit()` — автоматически поменяет ширину всех столбцов и высоту всех строк в диапазоне, чтобы туда уместился текст ячеек. Можно применять только к тем диапазонам, которые включают в себя хотя бы одну строку или столбец целиком, иначе вернется ошибка.
- `AutoFormat()` — позволяет использовать один из стилей автоформатирования, которые на графическом экране доступны через меню **Формат | Автоформат**.
- `BorderAround()` — позволяет поместить диапазон в рамку с выбранными вами параметрами.
- Методы с префиксом `Clear...` — позволяют очистить содержимое диапазона от значений, форматирования, комментарий и т. п.
- `Consolidate()` — сливает данные нескольких диапазонов (в том числе на разных листах) в один диапазон, используя при этом выбранную вами агрегатную функцию.
- `Copy()` — копирует диапазон. Если место назначения не указано, то он копируется в буфер обмена. Аналогично работает метод `Cut()`, при котором данные исходного диапазона удаляются.
- `CopyFromRecordset()` — очень удобный метод, который позволяет вставить данные из объекта `ADO.Recordset` на лист Excel, начиная с верхнего левого угла указанного диапазона.
- `DataSeries()` — метод, который поможет сэкономить множество времени и избежать возни с функциями даты и времени. Он позволяет увеличить

значения дат на указанный вами временной интервал. Например, если у вас в ячейке стоит 1 января, то при помощи этого метода можно сгенерировать первое число любого другого месяца.

- `Delete()` — удаляет данные текущего диапазона. С помощью необязательного параметра можно определить, с какой стороны будут сдвигаться ячейки на место удаленных.
- `Dirty()` — помечает ячейки диапазона как "грязные". Такие ячейки будут пересчитаны при следующем же пересчете. Этот метод обычно используется, когда Excel сам не может догадаться, что их нужно пересчитать. Также пересчитать ячейки диапазона можно и принудительно при помощи метода `Calculate()`.
- Методы с префиксом `Fill...` (`FillDown()`, `FillUp()`, `FillLeft()`, `FillRight()`) — позволяют размножить одно и то же значение по ячейкам диапазона в указанном вами направлении.
- `Find()` — позволяет произвести поиск по ячейкам диапазона и вернуть новый объект `Range`, который представляет собой первую ячейку, в которой было найдено нужное значение. У этого метода есть множество необязательных параметров, которые позволяют определить направление поиска, чувствительность к регистру, искать ли значение ячейки целиком или как часть и т. п. Методы `FindNext()` и `FindPrevious()` позволяют продолжить поиск, начатый методом `Find()`, в разных направлениях.
- `GoalSeek()` — позволяет применить автоподбор значений для функции Excel программным способом. На графическом экране то же самое можно сделать при помощи меню **Сервис | Подбор параметра**.
- `Insert()` — позволяет вставить ячейки в диапазон, сдвинув остальные вправо или вниз.
- `Justify()` — позволяет равномерно распределить текст по диапазону. Если в данный диапазон текст не помещается, то он будет распространен на соседние ячейки (с перезаписью их значений).
- `Merge()` — позволяет слить все ячейки диапазона в одну. При этом останется только одно значение — верхней левой ячейки. Разбить обратно такую слитую ячейку на несколько обычных можно при помощи метода `UnMerge()`.
- `Parse()` — позволяет разбить одну ячейку на несколько по указанному вами шаблону (например, чтобы отделить код города от номера телефона).
- `PasteSpecial()` — операция, дополняющая `Copy()` и `Cut()`. Она позволяет вставить то, что лежит в буфере обмена, с указанием специальных параметров вставки (вставлять с добавлением к существующим данным, с умножением, вычитанием, делением и т. п.).

- `PrintOut()` и `PrintPreview()` — позволяют вывести диапазон на печать или открыть режим просмотра перед печатью.
- `Replace()` — метод, дополняющий метод `Find()`. Позволяет проводить поиск и замену значений в диапазоне.
- `Select()` — выделяет указанный диапазон. Объекта `Selection` в Excel нет, вместо него есть возможность получить объект `Range`, представляющий выделенную область.
- `Show()` — экран будет пролистан таким образом, чтобы показать весь указанный диапазон.
- `ShowDependents()` — позволяет пометить стрелками те ячейки, которые зависят от указанного диапазона (только первый уровень зависимости) или убрать эти стрелки. Обратный метод — `ShowPrecedents()`.
- `ShowErrors()` — показывает источник ошибки для указанной ячейки.
- `Sort()` — производит сортировку ячеек в диапазоне. Можно использовать большое количество необязательных параметров для настройки сортировки. `SortSpecial()` — сортировка с учетом особенностей азиатских языков.
- `Speak()` — удивительный метод, который позволяет прочитать вслух содержимое диапазона (можно определить, в каком направлении и будут ли зачитываться формулы). К сожалению, в локализованной версии Excel не работает.
- `SpecialCells()` — очень удобный метод, который позволяет вернуть объект `Range`, включающий в себя все ячейки определенного типа (пустые, с ошибками, с комментариями, последние, с константами, с формулами, с определенным форматированием) и с определенным значением. Например, чтобы вернуть объект `Range`, состоящий из всех пустых ячеек диапазона, можно использовать код:

```
Set oRange2 = oRange.SpecialCells(xlCellTypeBlanks)
oRange2.Select    'Проверяем, так ли это
```

- `SubTotal()` — позволяет посчитать итоговое значение для диапазона (можно выбрать агрегатную функцию и множество других параметров).
- `Table()` — позволяет создать таблицу на основе передаваемого столбца, строки и функции, которую нужно использовать для вычисления ячеек таблицы. Пример в документации по этому методу позволяет автоматически сгенерировать таблицу умножения.
- `TextToColumns()` — сложный метод, который позволяет создать из значения одного столбца несколько столбцов в соответствии с определенным алгоритмом. Принимает множество необязательных параметров.

11.7. Коллекция *QueryTables* и объект *QueryTable*

Для большинства практических задач вполне хватает возможностей объектов `Application`, `Workbook`, `Worksheet` и `Range`. Например, для вставки информации из базы данных вы можете пройти циклом по объекту `ADO.Recordset` и вставить все нужные записи в лист Excel, а затем средствами VBA прописать в нижние строки итоги по вставленным данным. Однако в Excel встроено несколько специальных объектов, которые могут сильно упростить работу в различных ситуациях. Например, ту же операцию по вставке информации из базы данных удобнее будет провести при помощи специального объекта `QueryTable`, который будет рассматриваться в этом разделе. Еще два таких специальных объекта `PivotTable` и `Chart` будут рассматриваться в *разд. 11.8* и *11.9*.

Основное назначение объекта `QueryTable` — работа с набором значений, возвращаемых из базы данных. Этот объект доступен в Excel из графического интерфейса через меню **Данные | Импорт внешних данных | Импортировать данные**. При помощи объектов `QueryTable` вы можете разместить набор записей, полученных с источника данных, на листе Excel для выполнения с ним различных операций (например, анализа при помощи богатой библиотеки функций, для построения диаграмм, отчетов и т. п.). `QueryTable` удобно использовать для "односторонней" работы с источником данных, когда данные только скачиваются с источника в Excel, но изменять данные и сохранять изменения на источнике не нужно. В принципе, в Excel такую возможность синхронизации изменений реализовать можно (например, при помощи перехвата события `Change` объекта `Worksheet`), но намного проще и правильнее будет использовать для этой цели возможности Access. В этом разделе мы будем рассматривать только такую "однонаправленную" передачу данных из базы в Excel.

Как обычно, для того чтобы создать объект `QueryTable` и разместить его на листе, нужно использовать специальную коллекцию `QueryTables`, которая принадлежит рабочему листу (объекту `Worksheet`) и доступна через его одноименное свойство. Свойства и методы объекта `QueryTables` стандартные, как у большинства рассмотренных нами коллекций. Подробного рассмотрения заслуживает только метод `Add()`, при помощи которого и создается объект `QueryTable` (с одновременным добавлением в коллекцию). Этот метод принимает три параметра:

- `Connection` — источник данных для `QueryTable` (в виде объекта типа `Variant`). В качестве источника данных можно использовать:

- строку подключения OLE DB или ODBC *(см. гл. 9)*;
- готовый объект `Recordset`, созданный стандартными средствами ADO или DAO. При этом можно изменять `Recordset`, на который ссылается `QueryTable` и обновлять `QueryTable`. По многим причинам это самый удобный вариант при работе с `QueryTable`;
- другой объект `QueryTable` (вместе со строкой подключения и текстом запроса);
- текстовый файл;
- результаты Web-запроса или запроса Microsoft Query (в виде файла DQY или IQY). Создать такой файл запроса можно при помощи графических средств Excel в меню **Данные | Импорт внешних данных | Создать запрос**.

❒ `Destination` — куда вставлять полученную `QueryTable`. Передается объект `Range`, и вставка производится, начиная с верхнего левого угла диапазона.

❒ `SQL` — при помощи этого необязательного параметра можно определить SQL-запрос, который будет выполняться на источнике данных ODBC. Этот же запрос можно определить при помощи одноименного свойства объекта `QueryTable`.

Конечно, правильнее всего при создании `QueryTable` использовать готовый объект `Recordset`. В этом случае у нас будут и самые полные возможности настройки подключения и курсора, и возможность эффективного промежуточного хранения данных в оперативной памяти (в объекте `Recordset`), куда можно вносить изменения, и все удобные свойства и методы объекта `Recordset`. Код на создание объекта `QueryTable` на листе Excel может выглядеть так (мы используем тот же `Recordset` на основе таблицы `Northwind.Customers`, что и в *разд. 9.5*):

```
Dim cn As ADODB.Connection
Set cn = CreateObject("ADODB.Connection")
cn.Provider = "SQLOLEDB"
cn.ConnectionString = "User ID=SA;Password=password;" _
    & "Data Source = LONDON1;Initial Catalog = Northwind"
cn.Open
Dim rs As ADODB.Recordset
Set rs = CreateObject("ADODB.Recordset")
rs.Open "select * from dbo.customers", cn
Dim QT1 As QueryTable
Set QT1 = QueryTables.Add(rs, Range("A1"))
QT1.Refresh
```

Непосредственно помещение объекта `QueryTable` на лист производится при помощи метода `QueryTable.Refresh()`. Без него объект `QueryTable` будет создан только в оперативной памяти.

Информация о самых важных свойствах и методах объекта `QueryTable` представлена далее.

- `BackgroundQuery` — определяет, может ли выполнение запроса производиться в фоновом режиме, пока пользователь совершает в Excel другие действия. По умолчанию установлено в `True`. В `False` следует переводить только тогда, когда пользователь своими действиями в Excel может как-то помешать нормальной работе приложения.
- `CommandText` — текст команды SQL, т. е. текст запроса, который передается на источник. Существует совместно с аналогичным свойством `SQL` (которое оставлено для обратной совместимости) и имеет перед ним приоритет. При передаче `QueryTable` готового `Recordset` это свойство недоступно.
- `CommandType` — тип передаваемой в `CommandText` команды (вся таблица, SQL-запрос, имя куба OLAP и т. п.). При работе с готовым `Recordset` также недоступно.
- `Connection` — строка подключения, которую можно передать при вызове метода `Add()` коллекции `QueryTables`. Также при работе с готовым `Recordset` недоступно.
- `Destination` — второй параметр, который передается методу `Add()`. Возвращает объект `Range`, представляющий верхнюю левую ячейку диапазона, занимаемого на листе объектом `QueryTable`. После создания `QueryTable` доступен только для чтения.
- `EnableEditing` — определяет, может ли пользователь изменять на графическом экране свойства объекта `QueryTable`. По умолчанию установлено в `True`. Если перевести в `False`, то пользователь сможет только обновлять `QueryTable`.
- `EnableRefresh` — определяет, может ли пользователь обновлять `QueryTable`, получая заново данные (с источника или из `Recordset`).
- `FetchedRowOverflow` — это свойство принимает значение `True`, если записи, полученные с источника, не поместились на листе Excel (т. е. было скачано больше, чем 65 536 записей). Ошибки в такой ситуации не возникает, поэтому если вы работаете с большими наборами записей, имеет смысл реализовать соответствующие проверки.
- `FieldNames` — очень полезное свойство. Позволяет отключить вставку полученных с источника названий столбцов в первую строку `QueryTable`. По умолчанию `True` — вставлять названия столбцов.

- `MaintainConnection` — определяет, будет ли соединение с источником открыто все время до закрытия листа. По умолчанию `True` — оптимизировано для выполнения частых обновлений. Если переставить в `False`, можно сэкономить оперативную память на клиенте за счет уменьшения скорости обновления данных.
- `Name` — имя объекта `QueryTable` (на графическом экране его можно посмотреть, если на панели управления **Внешние данные** нажать кнопку **Свойства диапазона данных**). По умолчанию задается как `ExternalData_` *номер*.
- `Parameters` — позволяет получить доступ к коллекции `Parameters` (набор параметров запроса). Возможности практически такие же, как у параметров объекта `Recordset`.
- `PreserveColumnInfo` и `PreserveFormatting` — определяют, сохранять ли информацию о столбцах (сортировке, фильтрации и т. п.) и форматировании после обновления `QueryTable`. По умолчанию все сохраняется.
- `QueryType` — позволяет выяснить, что использовалось при создании `QueryTable`: `Recordset`, прямой доступ к таблице, SQL-запрос и т. п. Свойство доступно только для чтения.
- `Recordset` — возвращает ссылку на объект `Recordset`, который использовался для создания `QueryTable`, или позволяет заменить его для объекта `QueryTable` на другой (изменения вступят в силу только после вызова метода `Refresh()`).
- `Refreshing` — это свойство принимает значение `True` на момент выполнения фонового запроса к источнику. Если выполнение запроса слишком затянулось, его можно прервать при помощи метода `CancelRefresh()`.
- `RefreshOnFileOpen` — определяет, обновлять ли данные автоматически при открытии листа или можно обойтись уже скачанными значениями (по умолчанию).
- `RefreshPeriod` — определяет, через какие интервалы времени автоматически будет обновляться информация в `QueryTable` данными с источника. По умолчанию задан 0, т. е. автоматическое обновление отключено.
- `RefreshStyle` — определяет, что делать с существующими ячейками, на место которых вставляются ячейки `QueryTable` при обновлении.
- `ResultRange` — это, наверное, самое важное свойство объекта `QueryTable`. Как правило, данные из базы перекачиваются в Excel для дальнейшей обработки. Это свойство позволяет получить диапазон, который включает в себя все ячейки, вставленные на лист из объекта `QueryTable`, чтобы потом применить к ним различные функции (обычно по столбцам или по строкам). Чтобы этот метод сработал, обязательно нужно провести вставку

данных QueryTable на лист при помощи метода Refresh(). После этого можно будет использовать то, что возвращает это свойство, как обычный диапазон. Самый простой способ продемонстрировать работу этого метода — воспользоваться кодом:

```
QT1.ResultRange.Select
```

А следующий пример генерирует под первым столбцом QueryTable формулу с суммированием значений этого столбца:

```
Set c1 = Sheets("Лист1").QueryTables(1).ResultRange.Columns(1)
c1.Name = "Column1"
c1.End(xlDown).Offset(1, 0).Formula = "=SUM(Column1)"
```

- RowNumbers — свойство, которое может сильно упростить работу с данными, полученными при помощи QueryTable. Позволяет сгенерировать еще один столбец в QueryTable (слева), который будет состоять из номеров записей, полученных через QueryTable.
- SaveData — определяет, сохранять ли данные, полученные через QueryTable, вместе с книгой Excel. По умолчанию задано True. В False имеет смысл переводить это свойство для того, чтобы изначально гарантировать работу пользователя только с самыми последними данными, полученными из источника.
- SavePassword — определяет, сохранять ли пароль вместе со строкой подключения (это свойство можно использовать только для источников ODBC). Если переставить его в False, то можно повысить уровень безопасности вашего приложения.
- SourceDataFile — полный путь и имя файла источника (для Access, DBF и других настольных СУБД). Для клиент-серверных систем (таких как SQL Server) возвращает Null.
- Свойства с префиксом Text... — определяют параметры текстового файла, если этот файл был выбран в качестве источника для QueryTable.
- Свойства с префиксом Web... — определяют параметры данных, получаемых от запроса к Web-источнику.

Методы Refresh(), CancelRefresh() и Delete() объекта QueryTable очевидны и каких-либо комментариев не требуют. Метод ResetTimer() позволяет обнулить таймер автоматического обновления, а метод SaveAsODC() — сохранить определение источника данных в виде файла Microsoft Query (если источником был объект Recordset, то этот метод вернет ошибку).

У объекта QueryTable есть также два события: BeforeRefresh и AfterRefresh. Они срабатывают соответственно перед началом загрузки данных с источника и после окончания загрузки.

11.8. Работа со сводными таблицами (объект *PivotTable*)

В процессе работы большинства предприятий накапливаются так называемые необработанные данные *(raw data)* о деятельности. Например, для торгового предприятия могут накапливаться данные о продажах товаров (по каждой покупке отдельно), для предприятий сотовой связи — статистика нагрузки на базовые станции и т. п. Очень часто менеджменту предприятия необходима аналитическая информация, которая генерируется на основе необработанных данных, например, посчитать вклад каждого вида товара в доходы предприятия или качество обслуживания в зоне данной станции. Из необработанной информации такие сведения извлечь очень тяжело: нужно выполнять сложные SQL-запросы, которые обрабатываются долго и часто мешают текущей работе. Поэтому все чаще в настоящее время необработанные данные сводятся вначале в хранилище архивных данных Data Warehouse, а затем — в кубы OLAP, которые очень удобны для интерактивного анализа. Проще всего представить себе кубы OLAP как многомерные таблицы, в которых вместо стандартных двух измерений (столбцы и строки, как в обычных таблицах) измерений может быть очень много. Обычно для описания измерений в кубе используется термин "в разрезе". Например, отделу маркетинга может быть нужна информация во временном разрезе, в региональном разрезе, в разрезе типов продукта, в разрезе каналов продаж и т. п. При помощи кубов (в отличие от стандартных SQL-запросов) очень просто получать ответы на такие вопросы, как "сколько товаров такого-то типа было продано в четвертом квартале прошлого года в Северо-Западном регионе через региональных дистрибьюторов".

Конечно, в обычных базах данных такие кубы не создать. Для работы с кубами OLAP требуются специализированные программные продукты. Вместе с SQL Server поставляется база данных OLAP от Microsoft, которая называется Analysis Services. Есть OLAP-решения от Oracle, IBM, Sybase и других фирм.

Для работы с такими кубами в Excel встроен специальный клиент. По-русски он называется **Сводная таблица** (на графическом экране он доступен через меню **Данные | Сводная таблица**), а по-английски — **Pivot Table**. Соответственно, объект, который представляет этот клиент, называется `PivotTable`. Необходимо отметить, что он умеет работать не только с кубами OLAP, но и с обычными данными в таблицах Excel или в базах данных, но многие возможности при этом теряются.

Сводная таблица и объект `PivotTable` — это программные продукты фирмы Panorama Software, которые были приобретены Microsoft и интегрированы в Excel. Поэтому работа с объектом `PivotTable` несколько отличается от работы

с другими объектами Excel. Догадаться, что нужно сделать, часто бывает непросто. Поэтому рекомендуется для получения подсказок активно использовать макрорекордер. В то же время при работе со сводными таблицами пользователям часто приходится выполнять одни и те же повторяющиеся операции, поэтому автоматизация во многих ситуациях необходима.

При работе со сводной таблицей первое, что нам потребуется сделать, — это создать объект `PivotCache`, который будет представлять собой набор записей, полученных с источника OLAP. Очень условно этот объект `PivotCache` можно сравнить с `QueryTable`. Для каждого объекта `PivotTable` можно использовать только один объект `PivotCache`. Создание объекта `PivotCache` производится при помощи метода `Add()` коллекции `PivotCaches`:

```
Dim PC1 As PivotCache
Set PC1 = ActiveWorkbook.PivotCaches.Add(xlExternal)
```

`PivotCaches` — стандартная коллекция, из ее методов, которые заслуживают подробного рассмотрения, можно назвать только метод `Add()`. Этот метод принимает два параметра:

- `SourceType` — обязательный, определяет тип источника данных для сводной таблицы. Можно указать создание `PivotTable` на основе диапазона в Excel, данных из базы данных, во внешнем источнике данных, другой `PivotTable` и т. п. На практике OLAP имеет смысл использовать только тогда, когда данных много — нужно специализированное внешнее хранилище (например, Microsoft Analysis Services). В этой ситуации выбирается значение `xlExternal`.
- `SourceData` — обязательный во всех случаях, кроме тех, когда значение первого параметра равно `xlExternal`. Определяет тот диапазон данных, на основе которого и будет создаваться `PivotTable`. Обычно принимает в качестве значения объект `Range`.

Следующая задача — это настроить параметры объекта `PivotCache`. Как уже говорилось, этот объект очень напоминает `QueryTable`, и набор свойств и методов у него очень похожий. Некоторые наиболее важные свойства и методы представлены далее.

- `ADOConnection` — возвращает объект `ADO.Connection`, который автоматически создается при подключении к внешнему источнику данных. Используется для дополнительной настройки свойств подключения.
- `Connection` — работает точно так же, как и одноименное свойство объекта `QueryTable`. Может принимать строку подключения, готовый объект `Recordset`, текстовый файл, Web-запрос, файл Microsoft Query. Чаще всего при работе с OLAP строка подключения прописывается напрямую (по-

скольку получать объект Recordset, например, для изменения данных смысла нет — источники данных OLAP практически всегда доступны только для чтения). Настройка этого свойства для подключения к базе данных FoodMart (учебная база данных Analysis Services) на сервере LONDON может выглядеть так:

```
PC1.Connection = "OLEDB;Provider=MSOLAP.2;Data Source=LONDON1;" _
                 & "Initial Catalog = FoodMart 2000"
```

- CommandType и CommandText — описывают тип команды, которая передается на сервер баз данных, и текст самой команды. Например, чтобы обратиться к кубу Sales и получить его целиком в кэш на клиенте, можно использовать код типа:

```
PC1.CommandType = xlCmdCube
PC1.CommandText = Array("Sales")
```

- LocalConnection — позволяет подключиться к локальному кубу (файлу cub), созданному средствами Excel. Конечно, такие файлы для работы с "производственными" объемами данных использовать очень не рекомендуется — только для целей создания макетов и т. п.
- MemoryUsed — возвращает количество оперативной памяти, используемой PivotCache. Если PivotTable на основе этого PivotCache еще не создана и не открыта, то возвращается 0. Можно использовать для проверки, если ваше приложение будет работать на слабых клиентах.
- OLAP — возвращает True, если PivotCache подключен к серверу OLAP.
- OptimizeCache — позволяет оптимизировать структуру кэша. Изначальная загрузка данных будет производиться дольше, но потом скорость работы может существенно возрасти. Для источников OLE DB эта оптимизация не работает.

Остальные свойства объекта PivotCache совпадают с аналогичными свойствами объекта QueryTable, и поэтому здесь рассматриваться не будут.

Главный метод объекта PivotCache — это метод CreatePivotTable(). С его помощью и производится следующий этап — создание сводной таблицы (объекта PivotTable). Этот метод принимает четыре параметра:

- TableDestination — единственный обязательный параметр. Принимает объект Range, в верхний левый угол которого будет помещена сводная таблица.
- TableName — имя сводной таблицы. Если не указано, то автоматически сгенерируется имя типа "СводнаяТаблица1".

- `ReadData` — если установить в `True`, то все содержимое куба будет автоматически помещено в кэш. С этим параметром нужно быть очень осторожным, поскольку неправильное его применение может резко увеличить нагрузку на клиента.
- `DefaultVersion` — это свойство обычно не указывается. Позволяет определить версию создаваемой сводной таблицы. По умолчанию задается наиболее свежая версия.

Создание сводной таблицы в первой ячейке первого листа книги может выглядеть так:

```
PC1.CreatePivotTable Range("A1")
```

Сводная таблица у нас создана, однако сразу после создания она пуста. В ней предусмотрено четыре области, в которые можно размещать поля из источника:

- *область столбцов* — в нее помещаются те измерения ("разрез", в котором будут анализироваться данные), записей в которых меньше;
- *область строк* — те измерения, записей в которых больше;
- *область страницы* — те измерения, по которым нужно только проводить фильтрацию (например, показать данные только по такому-то региону или только за такой-то год);
- *область данных* — центральная часть таблицы. Те числовые данные (например, сумма продаж), которые мы будем анализировать.

Полагаться на пользователя в том, что он правильно разместит элементы во всех четырех областях, не стоит. Кроме того, это может занять определенное время. Поэтому часто требуется расположить данные в сводной таблице программным образом. Эта операция производится при помощи объекта `CubeField`. Главное свойство этого объекта — `Orientation`, оно определяет, где будет находиться то или иное поле. Например, помещаем измерение `Customers` в область столбцов:

```
PT1.CubeFields("[Customers]").Orientation = xlColumnField
```

измерение `Time` — в область строк:

```
PT1.CubeFields("[Time]").Orientation = xlRowField
```

измерение `Product` — в область страницы:

```
PT1.CubeFields("[Product]").Orientation = xlPageField
```

и, наконец, показатель `Unit Sales` (числовые данные для анализа):

```
PT1.CubeFields("[Measures].[Unit Sales]").Orientation = xlDataField
```

Теперь сводная таблица создана, и с ней можно работать. Однако часто необходимо выполнить еще одну операцию — раскрыть нужный уровень иерархии измерения. Например, если нас интересует поквартальный анализ, то нужно раскрыть уровень `Quarter` измерения `Time` (по умолчанию показывается только самый верхний уровень). Конечно, пользователь может сделать это самостоятельно, но не всегда можно рассчитывать, что он догадается, куда щелкнуть мышью. Программным образом раскрыть иерархию измерения `Time` на уровень кварталов для 1997 г. можно при помощи объектов `PivotField` и `PivotItem`:

```
PT1.PivotFields("[Time].[Year]").PivotItems("[Time].[1997]").
DrilledDown = True
```

11.9. Работа с диаграммами (объект *Chart*)

Одно из основных применений Excel — это анализ данных. А для анализа данных часто удобно использовать диаграммы с их специальными возможностями, такими как тренды. На практике задачи по автоматизации создания множества похожих друг на друга диаграмм (обычно на основе информации, полученной из базы данных) возникают очень часто.

С диаграммами в Excel существует некоторая терминологическая путаница. То, что на графическом интерфейсе русского Excel называется диаграммой (меню **Вставка | Диаграмма**), по-английски называется графиком *(Chart)* и ему соответствует объект `Chart`. В объектной модели Excel предусмотрен также и объект `Diagram`, но он представляет скорее схему отношений (то, что при помощи графического интерфейса русского Excel можно добавить при помощи меню **Вставка | Схематическая диаграмма**). Под диаграммой в этом разделе будет пониматься то же, что и у создателей русского Excel — обычный график.

Диаграммы в Excel создаются при помощи объекта `Chart`. Вначале лучше этот объект объявить:

```
Dim oChart As Chart
```

Дальше можно создавать диаграмму. Создание диаграммы производится при помощи много раз использованного нами приема — вызова метода `Add()` коллекции `Charts`:

```
Set oChart = ActiveWorkbook.Charts.Add(, ActiveSheet)
```

В принципе, диаграмма уже создана, но поскольку никакие ее свойства не определены, она выглядит как пустой лист. Чтобы она обрела содержание, необходимо выполнить еще несколько действий.

Первое (и единственное обязательное действие) — определить источник данных для диаграммы, для чего предназначен метод `SetSourceData()`. В качестве источника может выступать только объект `Range` (он передается в качестве первого и единственного обязательного параметра этого метода). Второй параметр (необязательный) определяет, в каком порядке считывать данные — сначала по столбцам, потом по строкам или наоборот. Например, в нашем случае это может выглядеть так:

```
oChart.SetSourceData Sheets("Лист1").Range("A1:A10")
```

В принципе, если запустить этот код на выполнение, то диаграмма уже будет создана. Для всех остальных параметров будут приняты значения по умолчанию. Однако на практике нужно определить еще хотя бы тип диаграммы (по умолчанию она будет выглядеть как "обычная гистограмма", т. е. ряд из столбиков разной длины). Для этой цели используется свойство `ChartType`, для которого разработчиками предусмотрено 73 значения. Например, чтобы преобразовать диаграмму в обычный график, можно использовать код вида:

```
oChart.ChartType = xlLineMarkers
```

Еще одна очень распространенная задача — добавить дополнительные ряды на диаграмму. Для этой цели необходимо создать и получить ссылку на объект `Series` — ряд, а потом для ряда определить свойство `Values` (ему передается в качестве значения объект `Range`):

```
Dim oSeries As Series
Set oSeries = oChart.SeriesCollection.NewSeries
oSeries.Values = Worksheets(1).Range("B1:B10")
```

Пользователи часто говорят, что им необходимо создавать диаграммы не на отдельном листе, а на том же листе, на котором расположены данные. По умолчанию диаграмма создается в оперативной памяти и помещается на отдельный лист. Если нам необходимо поместить ее на уже существующий лист, то в этом случае ее надо создать вначале на отдельном листе, а затем переместить при помощи метода `Location`. Отдельный лист, созданный для диаграммы, при этом автоматически исчезнет:

```
oChart.Location xlLocationAsObject, "Лист1"
```

Обратите внимание, что метод `Location` принимает в качестве первого параметра одну из констант (`xlLocationAsNewSheet` — переместить на специально создаваемый новый лист, `xlLocationAsObject` — переместить на объект, т. е. на лист), а в качестве второго — не объект листа, а его имя. Если код предполагается использовать и в русской, и в английской версии Excel, то предпочтительнее получать имя листа программным образом.

Большая неприятность, связанная с методом `Location`, заключается в том, что после перемещения диаграммы внутрь листа объектная ссылка на эту диа-

грамму теряется, и надо находить объект этой диаграммы заново. При попытке повторного обращения к объекту `Chart` выдается сообщение `"Automation Error"`. Поэтому лучше всего вызов метода `Location` помещать в самый конец кода, посвященного диаграмме. В противном случае нам придется разыскивать созданную нами диаграмму и заново получать на нее объектную ссылку, например, так:

```
Dim oSeries As Series
Set oSeries = _
    Worksheets(1).ChartObjects(1).Chart.SeriesCollection.NewSeries
oSeries.Values = Worksheets(1).Range("B1:B10")
```

Так работать, конечно, очень неудобно.

Остальные многочисленные параметры диаграммы настраиваются при помощи свойств и методов объектов `Chart`.

- `ChartArea` — это свойство возвращает одноименный объект `ChartArea`, который представляет собой область, занимаемую диаграммой, и используется для настройки внешнего вида диаграммы (свойства `Font`, `Interior` и т. п.). Если необходимо настроить внешний вид не всей диаграммы, а той ее части, которая используется непосредственно для вывода графика, используется схожее свойство `PlotArea`. По умолчанию диаграмма размещается прямо по центру листа. Если необходимо ее переместить в точно определенное место листа, используются знакомые свойства `Top`, `Height`, `Left` и `Width` объекта `ChartArea`.
- `ChartTitle` — возвращает одноименный объект, при помощи которого можно настроить заголовок диаграммы (с такими свойствами, как `Text`, `Font`, `Border` и т. п.).
- `ChartType` — важнейшее свойство, про которое мы уже говорили. Определяет тип диаграммы.
- `HasDataTable` — если установить это свойство в `True`, то в нижней части диаграммы (по умолчанию) появится таблица с числами, на основе которых была создана диаграмма. Одновременно будет создан программный объект `DataTable`, при помощи которого можно будет настроить представление этой таблицы. Схожим образом работают свойства `HasLegend`, `HasPivotFields` и `HasTitle`.
- `Name` — это свойство позволяет настроить имя диаграммы (как название вкладки в Excel). По умолчанию диаграммы называются последовательно `"Диаграмма1"`, `"Диаграмма2"` и т. п.
- `SizeWithWindow` — если поставить значение этого свойства в `True`, то размер диаграммы будет подогнан таким образом, чтобы точно соответствовать размеру листа. По умолчанию установлено в `False`.

- `Tab` — свойство, о котором мало кто знает. Оно позволяет настроить при помощи одноименного объекта внешний вид вкладки в книге Excel для диаграммы (или просто листа). Например, чтобы пометить вкладку зеленым цветом, можно воспользоваться кодом:

```
oChart.Tab.Color = RGB(0, 255, 0)
```

- `Visible` — позволяет спрятать диаграмму без ее удаления.

Остальные свойства в основном относятся к настройке отображения трехмерных диаграмм и к защите диаграммы от изменения пользователем.

Далее представлены самые важные методы объекта `Chart`.

- `Activate()` — используется очень часто. Он позволяет сделать диаграмму активной (т. е. просто перейти на нее).
- `ApplyCustomType()` — позволяет создать диаграмму своего собственного пользовательского типа (для этого необходимо вначале создать шаблон для этого типа и поместить его в галерею).
- `ApplyDataLabels()` — позволяет поместить на диаграмму метки для размещенных на ней данных. Этот метод принимает множество параметров, которые позволяют настроить отображение меток (показывать или не показывать значения и т. п.).
- `Axes()` — возвращает объект, представляющий оси диаграммы. Затем этот объект можно использовать для настройки данных осей.
- `ChartWizard()` — этот метод позволяет быстро переформатировать диаграмму, как если бы вы прошли на графическом экране Мастер построения диаграмм и передали ему значения. Позволяет при помощи одной строки кода добиться того, что другими способами заняло бы несколько строк.
- `Copy()` — позволяет скопировать диаграмму в другое место книги (например, для создания новой диаграммы на основе существующей). Для переноса существующей диаграммы в другое место можно воспользоваться методами `Location()` или `Move()`.
- `CopyPicture()` — замечательный метод, который позволяет поместить диаграмму в буфер обмена как изображение. Затем это изображение можно вставить, например, в документ Word или в любое другое место. Еще один вариант — воспользоваться методом `Export()`, который позволяет создать рисунок, представляющий диаграмму, в виде файла на диске.
- `Delete()` — удаляет диаграмму.
- `Evaluate()` — как обычно, этот метод позволяет найти нужную диаграмму в книге по ее имени.

- `PrintOut()` — отправляет диаграмму на печать. Этот метод принимает множество параметров, которые позволяют настроить такой вывод.
- `Refresh()` — позволяет обновить диаграмму, если изменились данные, на основе которых она строилась.
- `Select()` — выделяет диаграмму (равносильно щелчку по ней левой кнопкой мыши). Обратный метод `Deselect()` снимет выделение (равносильно нажатию <Esc>).
- `SetBackgroundPicture()` — позволяет "подложить" под диаграмму фоновый рисунок. Конечно, он должен быть не очень ярким.
- `SetSourceData()` — важнейший метод, который позволяет определить данные, на основе которых строится диаграмма. Про него мы уже говорили.

Для объекта `Chart` предусмотрено также события "на все случаи жизни" — реакции на щелчки мышью, на выделение или снятие выделения, активизацию, пересчет данных, изменение размера и т. п., однако на практике такие события используются редко.

11.10. Другие объекты Excel

В Excel предусмотрено множество других объектов, подробно рассмотреть которые не позволяет объем данной книги. Далее представлен обзор самых важных из них. Найти более подробную информацию о работе с ними можно при помощи официальной документации или макрорекордера.

Объект `CellFormat` позволяет настраивать форматирование ячеек Excel — шрифт, рамки, цветовое оформление и т. п. При этом можно использовать этот объект для копирования оформления с других ячеек, для поиска и замены по оформлению и т. п. Условное форматирование можно настроить при помощи объекта `FormatCondition`.

`ListObject` — новый объект (появился только в Office 2003), который предназначен для работы со списками — наборами взаимосвязанных данных (например, списки сотрудников). Обычно используется при работе с книгами Excel на SharePoint Portal Server.

Объект `Validation` позволяет настроить проверку вводимых пользователем данных.

Объект `Watch` позволяет настроить контрольное значение для формул. При помощи контрольного значения (меню **Сервис | Макрос | Показать контрольное значение**) можно отслеживать значения тех формул, которые находятся за пределами экрана.

Задание для самостоятельной работы 11: Применение Excel для анализа информации из базы данных

ЗАДАНИЕ:

В вашей компании ведется учет товаров, которые имеются на складе, при помощи таблицы `Товары` базы данных `Борей`, которая расположена в каталоге C:\Program Files\Microsoft Office\OFFICE11\SAMPLES. В этой таблице находятся следующие важные для вас столбцы:

- `КодТовара` — идентификатор товара;
- `Марка` — наименование продукта;
- `Цена` — стоимость за единицу продукта;
- `НаСкладе` — количество единиц этого товара на складе;
- `МинимальныйЗапас` — минимально допустимое количество единиц данного товара на складе. Если реальное количество единиц этого товара меньше, чем это значение, то товар нужно срочно заказать;
- `ПоставкиПрекращены` — флаг прекращения работы с товаром. Если в этом столбце стоит единица, то это значит, что принято решение закупки этого товара больше не производить.

Все остальные столбцы для целей этой работы можно игнорировать.

Заполнение таблицы `Товары` производится при помощи специализированного приложения, созданного достаточно давно и не предусматривающего некоторых необходимых форм.

Вам поручено создать приложение на основе Excel, которое бы:

1. Производило вставку в лист Excel данных по всем строкам и всем столбцам этой таблицы.
2. Генерировало бы в Excel дополнительные столбцы следующего содержания:
   - `"Заказать товара, штук"` — разница между столбцами `МинимальныйЗапас` и `НаСкладе`. В этот столбец должна помещаться информация о количестве товара в штуках, которое нужно срочно заказать. Эту информацию нужно генерировать только для тех записей, для которых значение в столбце `МинимальныйЗапас` больше, чем в столбце `НаСкладе`, и у которых значение столбца `ПоставкиПрекращены` установлено в `False`;
   - `"Стоимость заказа"` — определяло бы стоимость такого пополнения склада для каждой строки в таблице. Стоимость заказа рассчитывается как произведение предыдущего столбца и столбца `Цена`. Эту информа-

цию также нужно генерировать только для тех записей, для которых значение в столбце МинимальныйЗапас больше, чем в столбце НаСкладе.

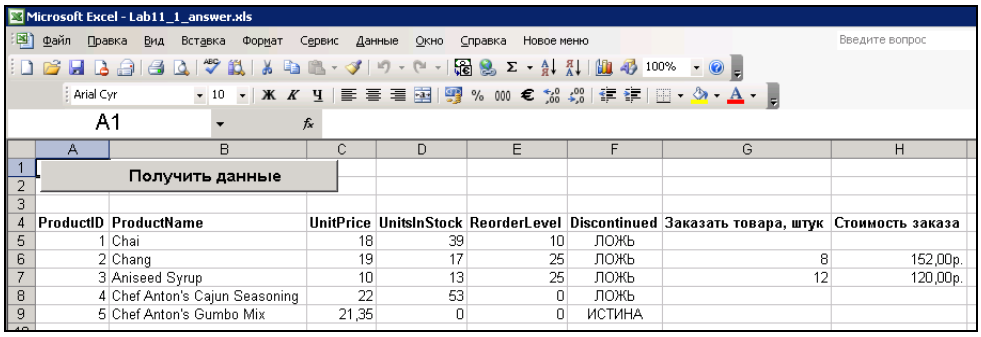
Примечание

В реальной задаче правильнее (и намного производительнее) бы было перенести расчет таких столбцов на сервер баз данных, использовав SQL-запрос с вычисляемыми столбцами. Однако для целей этой самостоятельной работы реализуйте их вставку средствами Excel (такое решение — единственно возможное, к примеру, если мы обращаемся к нереляционному источнику данных, такому как текстовый файл).

3. Вставляло бы в одной строке под полученными записями из базы данных две итоговые строки:
   - "Общая стоимость товаров на складе" — итоговая стоимость всех товаров, которые находятся на складе (как сумма произведений столбцов НаСкладе и Цена для каждой строки);
   - "Общая стоимость товаров к заказу" — итог по столбцу СтоимостьЗаказа.

Общий вид получившегося приложения может быть таким, как представлено на рис. 11.2.

Итоговые строки могут выглядеть так, как показано на рис. 11.3.



	ProductID	ProductName	UnitPrice	UnitsInStock	ReorderLevel	Discontinued	Заказать товара, штук	Стоимость заказа
5	1	Chai	18	39	10	ЛОЖЬ		
6	2	Chang	19	17	25	ЛОЖЬ	8	152,00р.
7	3	Aniseed Syrup	10	13	25	ЛОЖЬ	12	120,00р.
8	4	Chef Anton's Cajun Seasoning	22	53	0	ЛОЖЬ		
9	5	Chef Anton's Gumbo Mix	21,35	0	0	ИСТИНА		

Рис. 11.2. Первые строки листа с импортированными данными

73	69	Gudbrandsdalsost	36	26	15	ЛОЖЬ		
74	70	Outback Lager	15	15	30	ЛОЖЬ	15	225,00р.
75	71	Flotemysost	21,5	26	0	ЛОЖЬ		
76	72	Mozzarella di Giovanni	34,8	14	0	ЛОЖЬ		
77	73	Röd Kaviar	15	101	5	ЛОЖЬ		
78	74	Longlife Tofu	10	4	5	ЛОЖЬ	1	10,00р.
79	75	Rhönbräu Klosterbier	7,75	125	25	ЛОЖЬ		
80	76	Lakkalikööri	18	57	20	ЛОЖЬ		
81	77	Original Frankfurter grüne Soße	13	32	15	ЛОЖЬ		
82								
83								
84		**Общая стоимость товаров на складе:**	**74 050,85р.**					
85		**Общая стоимость товаров к заказу:**	**3 633,45р.**					
86								

Рис. 11.3. Последние строки с итоговыми значениями

Ответ к заданию 11

1. Создайте новый файл Excel, сделайте видимым панель управления **Элементы управления**, щелкните в нем по элементу управления **Кнопка** и поместите кнопку на лист Excel. Для наших целей мы будем считать, что созданная кнопка занимает две верхние строки первого листа.
2. На панели инструментов **Элементы управления** щелкните по кнопке **Свойства** (при этом созданная кнопка должна быть выделена) и настройте для свойства `Caption` значение `"Получить данные"`. Воспользуйтесь свойством `Font`, чтобы настроить подходящий шрифт для вашей кнопки.
3. Щелкните правой кнопкой мыши по созданной вами кнопке и в контекстном меню выберите **Исходный текст**. Откроется редактор Visual Basic с курсором ввода на месте события `Click` для вашей кнопки.
4. В окне редактора кода в меню **Tools** выберите **References** и установите флажок напротив строки **Microsoft ActiveX Data Objects 2.1 Library**.
5. Код для события `Click` вашей кнопки **Получить данные** может быть таким, как показано далее.

```
Private Sub CommandButton1_Click()
    'Вначале — чистим всю книгу от старых данных
    Cells.Select
    Selection.Clear

    'Создаем и настраиваем объект Connection
    Dim cn As New ADODB.Connection
    cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
& "Data Source=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Борей.mdb"
    cn.Open

    'Создаем и настраиваем объект Recordset
    Dim rs As New ADODB.Recordset
    rs.Open "SELECT [КодТовара], [Марка], [Цена], [НаСкладе]," _
        & "[МинимальныйЗапас], [ПоставкиПрекращены] FROM Товары", cn

    'На основе Recordset создаем объект QueryTable и
    'вставляем его, начиная с 4-й строки
    Dim QT1 As QueryTable
    Set QT1 = QueryTables.Add(rs, Range("A4"))
    QT1.Refresh

    'Определяем количество записей в QueryTable
    Dim nRowCount As Integer
    Dim oRange As Range
```

```
Set oRange = QT1.ResultRange
nRowCount = oRange.Rows.Count

'Формируем столбец "Заказать товара, штук"
Range("G4").Value = "Заказать товара, штук"
Range("G4").Font.Bold = True
Range("G4").Columns.AutoFit

'Формируем столбец "Стоимость заказа"
Range("H4").Value = "Стоимость заказа"
Range("H4").Font.Bold = True
Range("H4").Columns.AutoFit

'Создаем диапазон, который включит в себя столбец G
' "вдоль" QueryTable
Set oRange = Range("G5", "G" & nRowCount + 3)

'Готовим переменные, которые нам потребуются в цикле
Dim oCell As Range
Dim sRowNumber As String
Dim cMoney As Currency
Dim cItogMoney As Currency
Dim cItogSklad As Currency

'Проходим циклом по всем ячейкам созданного диапазона
For Each oCell In oRange.Cells
    'Получаем абсолютный номер строки в виде строковой переменной
    sRowNumber = Replace(oCell.Address(True), "$G$", "")
    'Проверяем определенные нами условия
    If Range("E" & sRowNumber).Value > Range("D" & sRowNumber) And _
        Range("F" & sRowNumber).Value = False Then

        'Получаем значение для столбца G (заказ в штуках)
        oCell.Value = (CInt(Range("E" & sRowNumber).Value) - _
                       CInt(Range("D" & sRowNumber).Value))

        'Получаем значение для столбца H (стоимость заказа)
        cMoney = (CInt(Range("E" & sRowNumber).Value) - _
                  CInt(Range("D" & sRowNumber).Value)) * _
                  CCur(Range("C" & sRowNumber).Value)
        'Записываем его в столбец H
        Range("H" & sRowNumber).Value = cMoney
        'Сразу плюсуем к итогу в рублях
        cItogMoney = cItogMoney + cMoney
    End If
```

```
        'И в том же цикле сразу суммируем стоимость товаров на складе
        cItogSklad = cItogSklad + (Range("C" & sRowNumber).Value * _
                    Range("D" & sRowNumber).Value)

    Next

    'Формируем две строки с итогами
Range("B" & nRowCount + 6).Value = "Общая стоимость товаров на складе:"
Range("B" & nRowCount + 6).Font.Bold = True
Range("B" & nRowCount + 7).Value = "Общая стоимость товаров к заказу:"
    Range("B" & nRowCount + 7).Font.Bold = True
    Range("D" & nRowCount + 6).Value = cItogSklad
    Range("D" & nRowCount + 6).Font.Bold = True
    Range("D" & nRowCount + 7).Value = cItogMoney
    Range("D" & nRowCount + 7).Font.Bold = True

    'Для красоты выделяем итоговое значение ...
    Range("D" & nRowCount + 7).Select
    '... и производим скроллирование
    Range("D" & nRowCount + 7).Show

End Sub
```