Глава 3

Написание структурированных VBA-программ

В этой главе...

- Структура большинства программ
- Преимущества структурирования
- Создание программ путем записи макросов
- Создание программ с использованием подпрограмм
- Создание программ с использованием функций
- Область видимости переменных
- Повышение наглядности программы
- Добавление комментариев

В главах 1 и 2 я сконцентрировался на описании принципов программирования на VBA, не рассмотрев важный элемент VBA-программ — их структуру. Благодаря структурированию программный код становится более легким для чтения и использования. Кроме того, структурирование программ — обязательная часть процесса разработки.

Существует множество подходов к структурированию. Подобно тому, как выстраивается схема презентации, в программу привносится структура, чтобы обеспечить ее правильную работу. Это напоминает использование контрольного списка для большого клиентского заказа — необходимо удостовериться в том, что все элементы находятся на своих местах. Четкая структура облегчает восприятие и понимание программы.

В этой главе будут представлены различные формы структуры. Наиболее наглядный структурный подход — физическое разделение. Программу можно разделить на части, удобные для написания и понимания. Как правило, отдельные части ориентированы на решение конкретных задач. При использовании средства записи макросов (называемого также *макрорегистратором*) можно наглядно увидеть, как программируются небольшие задачи, поэтому в настоящей главе мы познакомимся с ним поближе.

Еще одна форма структурирования включает понятие конфиденциальности. Подумайте над тем, кому разрешено просматривать программу и работать с ней. Важное значение имеет так называемая *область видимости*, которая представляет собой не что иное, как определение уровней доступа к программе. Вам может потребоваться обеспечить полную конфиденциальность некоторых частей программы, а другие части сделать полностью открытыми.

Наконец, существуют визуальные элементы структурирования. Наличие (или отсутствие) пробелов и пустых строк определяет, будет ли программа легкой для чтения или же заставит пользователя в недоумении чесать затылок. В главе 2 был рассмотрен вопрос использования комментариев в форме псевдокода, но могут потребоваться и дополнительные комментарии, помогающие другим людям разобраться в вашей программе.

Части программы

Примеры в главах 1 и 2 имели пусть очень простую, зато достаточно четкую структуру. VBA требуется эта структура, чтобы понять, в какой последовательности выполнять действия. Пользователь может пренебречь структурой, а VBA — нет. В этой главе объясняется формальный смысл каждого структурного элемента.

Определение частей программы

Программа — наивысший уровень физического структурирования. Она содержит все компоненты, необходимые для выполнения задачи. Программа может пересекать границы модуля, модуля класса и формы. Это специальные контейнеры для хранения программного кода. Вы можете создавать их в виде отдельных файлов, чтобы обращаться к ним впоследствии, но приложения Office встраивают их внутрь документа или шаблона. Программа служит контейнером кода, используемого для реализации набора функций, которые требуются операционной системе или пользователю.

Некоторым людям трудно понять, что такое программа, поскольку современные программные пакеты зачастую определяют этот термин некорректно. Когда вы запускаете Word, это *программа*. С другой стороны, Microsoft Office — это набор программ, или *пакет приложений*. Microsoft Office — это не одна программа, а целая их группа, включающая Word, PowerPoint, Excel и другие приложения.

Драйвер отдельного устройства (скажем, драйвер, который вы устанавливаете для мыши) — это программа. Интерфейс пользователя, который помогает настроить конфигурацию мыши, тоже зачастую представляет собой отдельную программу. Это не часть драйвера устройства, хотя и позволяет управлять его работой.

Не следует полагать, будто программа и проект — это одно и то же. *Проект* — это контейнер, используемый для хранения встроенных модулей, модулей классов и форм, связанных с данным документом. Создавая новый проект, вы не создаете новую программу: проект VBA может в действительности включать несколько программ.

Любая общедоступная подпрограмма типа `Sub`, к которой может получить доступ пользователь посредством диалогового окна Макрос (как описывалось в главе 2), — это отдельная программа. Макрос `SayHello`, создававшийся в главе 2, может служить примером простейшей программы, в основе которой лежит одна процедура `Sub`, но равно можно создавать и программы любой сложности.

В главе 16 будет показано, что отдельная программа может выходить за границы проекта. Все примеры в той главе основаны на функциях, предоставляемых по меньшей мере двумя приложениями Microsoft Office, но направленных на решение одной задачи. Несмотря на то что программа обращается к функциям нескольких приложений, это по-прежнему одна программа. Физическое размещение и использование внешних модулей никак не влияет на это определение.

Строительные блоки VBA-программ

VBA-программа состоит из строительных блоков. Подобная аналогия характерна для объяснения принципов работы программ. Если не понимать абстрактных элементов программирования на VBA, то будет невозможно писать программы. Ниже приведены описания важнейших конструктивных элементов программы.

- ✓ **Проект.** Проект служит контейнером для модулей, модулей классов и форм в рамках конкретного файла. В Word обычно присутствует как минимум три проекта, за-

груженных в редактор Visual Basic: шаблон `Normal`, шаблон документа и сам документ. Пользователи Excel видят только один проект — текущего открытого файла.

- ✓ **Модуль, модуль класса и форма.** Эти три элемента служат контейнерами основных программных конструкций, таких как описания классов и процедур. Отдельный проект может включать несколько модулей, модулей классов и форм, однако каждый из этих элементов должен обладать уникальным именем.
- ✓ **Функция (`Function`) и подпрограмма (`Sub`)**. Блоки `Function` и `Sub` хранят отдельные строки кода (также называемые *инструкциями*). Функция возвращает значение вызывающей программе, подпрограмма — нет. Пакет Microsoft Office обеспечивает доступ к функциональным возможностям кода посредством подпрограмм, а не функций. Следовательно, вы всегда должны использовать блоки `Sub`.
- ✓ **Инструкция.** Многие люди называют отдельную строку кода *инструкцией*. Из примера псевдокода в главе 2 понятно почему. Каждая строка псевдокода говорит о том, какие именно операции должно выполнить приложение. Пример в разделе "Этап 2. Реализация проекта" главы 2 показывает, как эти строки псевдокода транслируются в программный код, понятный VBA. Инструкция — это просто предложение, но не на обычном языке, а на языке VBA.

Использование макрорегистратора

Макрорегистратор (средство записи макросов) позволяет зафиксировать нажатия клавиш и выполняемые операции в виде VBA-программы. Его можно использовать для записи сложных процедур, таких как настройка документа, или вспомогательных действий, таких как выделение текста или изменение параметров форматирования. Макрорегистратор может помочь в выполнении следующих задач:

- ✓ создать макрос на основе действий пользователя;
- ✓ определить, как в приложении реализуются те или иные действия;
- ✓ разделить программу на процедуры;
- ✓ подготовить основу для более сложной программы.

Макрорегистратор не решает всех проблем. Например, не получится использовать макрорегистратор для создания интерактивных программ, не прибегнув к дополнительному программированию. То же справедливо и в отношении программ, которые должны изменяться в зависимости от пользовательского ввода, системной среды или данных, с которыми выполняются манипуляции. Все эти задачи требуют написания дополнительного кода. Однако макрорегистратор — хорошая отправная точка для многих задач структурного программирования. С его помощью можно быстро создать основу программы, а затем внести необходимые изменения. Независимо от используемой версии Office, процесс записи макросов состоит из следующих основных этапов.

1. Запуск макрорегистратора.
2. Последовательное выполнение всех действий, которые обычно требуются для решения задачи.
3. Останов макрорегистратора.
4. Сохранение макроса после того, как приложение предложит вам это сделать.
5. Дополнительно: открытие записанного макроса и внесение всех необходимых изменений.

Запись макроса с использованием интерфейса ленты

Записать макрос с помощью новых средств, предоставляемых в распоряжение интерфейсом ленты Office 2007, легче, чем в предыдущих версиях. Компания Microsoft добавила в свои приложения функциональные возможности, которые упрощают процесс создания макросов. Например, при нажатии клавиши <Alt> в маленьком окошке над каждым элементом управления на ленте отображается цифра или буква, которую следует нажать для выполнения определенного действия.

Если вы ранее постоянно использовали мышь для выполнения большинства задач, вам может потребоваться несколько раз попрактиковаться в последовательном нажатии клавиш, необходимых для записи макроса. Чем меньше ошибок сделано при записи макроса, тем легче впоследствии его редактировать. Ниже описывается процесс записи макроса с использованием интерфейса ленты.

1. **Выберите вкладку Разработчик (рис. 3.1).**

   Поскольку Office 2007 не отображает эту вкладку по умолчанию, подробные инструкции по ее отображению можно найти в главе 1.

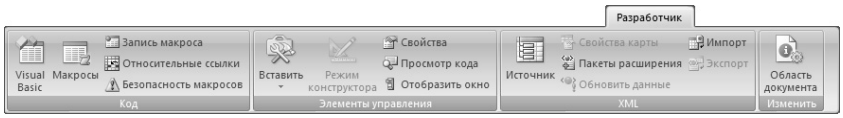

Рис. 3.1. Вкладка Разработчик включает большинство компонентов, необходимых для работы с макросами

2. **Щелкните на кнопке Запись макроса.**

   Откроется диалоговое окно Запись макроса, показанное на рис. 3.2.

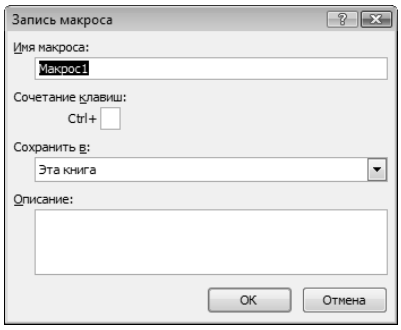

Рис. 3.2. Для ввода информации, касающейся вашего макроса, используйте диалоговое окно Запись макроса

3. **Введите описательное имя для макроса.**
4. **Если хотите вызывать макрос с помощью клавиатуры, задайте для него комбинацию клавиш.**

   Эту возможность следует использовать только для основных макросов, поскольку количество доступных комбинаций клавиш ограничено.

5. **Выберите место хранения макроса в поле Сохранить в.**

Место хранения зависит от приложения. Ниже приведены пояснения применительно к программе Excel.

- **Эта книга.** Этот вариант следует использовать в том случае, если вы намерены хранить макрос в локальном файле. Любой, кто откроет файл, получит доступ к макросу.
- **Личная книга макросов.** Используйте этот вариант, если хотите хранить макрос в специальной рабочей книге, содержащей все ваши личные макросы. Это место хранения делает этот макрос постоянно доступным для вас. При этом не имеет значения, какую рабочую книгу вы открыли.
- **Новая книга.** Этот вариант следует использовать, если вы хотите хранить макрос в новой рабочей книге.

Пояснения для программы Word приведены ниже.

- **Документ.** Этот вариант следует использовать в том случае, если вы намерены хранить макрос в локальном файле. Любой, кто откроет файл, получит доступ к макросу.
- **Документов, основанных на шаблоне.** Используйте этот вариант для хранения макроса в шаблоне, подключаемом к документу. Любой, кто создает документ, основанный на данном шаблоне, получает доступ к макросу.
- **Всех документов (Normal.dotm).** Используйте этот вариант для хранения макроса в глобальном шаблоне. Хранение макроса в этом месте означает, что любой, кто откроет документ любого рода, получит доступ к макросу.

6. **Введите описание макроса в поле Описание.**

   Важно ввести полное описание макроса, поскольку это единственный комментарий, который будет содержать макрос после завершения его создания.

7. **Щелкните на кнопке OK.**

   Приложение начнет запись макроса. Обратите внимание, что кнопка Запись макроса заменяется кнопкой Остановить запись, а пиктограмма кнопки меняет красный цвет на голубой.

8. **Выполните необходимые действия.**

   Приложение записывает все нажатия клавиш. Однако оно не записывает перемещения указателя мыши. Следовательно, вам нужно избегать использования мыши и выполнять все задачи с помощью клавиатуры.

9. **Щелкните на кнопке Остановить запись.**

   Приложение завершает запись макроса.

Вы можете просмотреть новый макрос, щелкнув на кнопке Макрос вкладки Разработчик. Диалоговое окно Макрос отображает макрос, связанный с текущим документом, независимо от того, является ли он локальным файлом, внешним документом или шаблоном. В разделе “Изменение макроса” это диалоговое окно будет описано более подробно.

Запись макроса с использованием интерфейса меню

Более ранние версии Office и некоторые приложения пакета Office 2007 требуют использования интерфейса меню для активизации макрорегистратора. Ниже описана последовательность действий для записи макроса с использованием меню.

1. **Выберите команду Сервис⇨Макрос⇨Начать запись.**

   Откроется диалоговое окно Запись макроса (см. рис. 3.2).

2. **Введите описательное имя макроса.**
3. **Введите комбинацию клавиш для макроса, если хотите вызывать его с помощью клавиатуры.**

   Эту возможность следует использовать только для основных макросов, поскольку количество доступных комбинаций клавиш ограничено.
4. **Выберите место хранения макроса в поле Сохранить в.**

   (Эта возможность присутствует только в Office 2007; в более старых версиях макросы всегда хранятся в локальном документе.)

   Место хранения зависит от приложения. Ниже приведены пояснения применительно к программе Visio.

   - **Active Document.** Эту опцию следует использовать в том случае, если вы намерены хранить макрос в локальном файле. Любой, кто откроет файл, получит доступ к макросу.
   - **Stencil.** Макрос хранится в файле шаблона. Любой, кто использует этот шаблон, получает доступ к макросу, независимо от того, какой документ открыт.
5. **Введите описание макроса в поле Описание.**

   Важно ввести полное описание макроса, поскольку это единственный комментарий, который будет содержать макрос после завершения его создания.
6. **Щелкните на кнопке OK.**

   Приложение начнет запись макроса. Отображается панель останова записи, показанная на рис. 3.3. Эта панель включает кнопки останова и паузы. Пауза в записи позволяет выполнить изменения, которые не требуется включать в макрос. Чтобы возобновить запись, щелкните на кнопке Пауза еще раз.
7. **Выполните необходимые действия.**

   Приложение записывает все нажатия клавиш. Однако оно не записывает перемещения указателя мыши. Следовательно, нужно избегать использования мыши и выполнять все задачи с помощью клавиатуры.
8. **Щелкните на кнопке Остановить запись.**

   Приложение завершает запись макроса. Панель останова записи исчезает с экрана.

Рис. 3.3. При необходимости можно остановить процесс записи макроса или сделать паузу

Можно просмотреть новый макрос, выбрав команду меню Сервис⇨Макрос⇨Макросы. Диалоговое окно Макрос отображает макрос, связанный с текущим документом, независимо от того, является ли он локальным документом, частью другого документа, шаблоном или другим связанным файлом.

Изменение макроса

Макрос, записанный с помощью макрорегистратора, редактируется так же, как и любой другой макрос. Единственное различие состоит в том, что вы не писали исходный программ-

ный код сами, а макрорегистратор не добавил в текст макроса никаких комментариев. Для примера откройте в Excel созданный выше макрос. Ниже описано, как это сделать.

1. **Откройте диалоговое окно Макрос. При использовании интерфейса ленты щелкните на кнопке Макросы вкладки Разработчик. При использовании меню выберите команду Сервис⇨Макрос⇨Макросы.**

   Отображается диалоговое окно Макрос, показанное на рис. 3.4.

2. **Выберите макрос, который намерены изменить, а затем щелкните на кнопке Изменить.**

   Приложение запустит редактор Visual Basic, в котором будет открыт выбранный макрос, как показано на рис. 3.5. (Вы можете увидеть и другие открытые файлы макросов.)

3. **Добавьте комментарии к записанному макросу, чтобы впоследствии можно было быстро вспомнить назначение команд.**

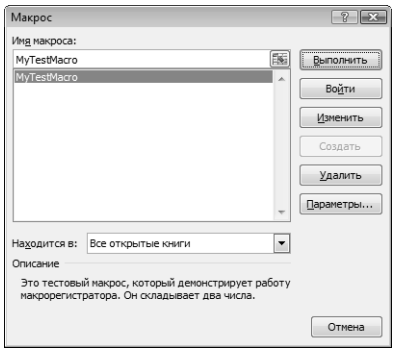

Рис. 3.4. Диалоговое окно Макрос содержит список доступных макросов

4. **Внесите необходимые изменения в макрос.**
5. **Сохраните макрос и закройте редактор Visual Basic.**

Пока нет необходимости вникать в суть макроса, показанного на рис. 3.5. Тем не менее дам краткие пояснения. Работа макроса начинается с присвоения значения `1` ячейке `A1` рабочего листа. Поскольку в начале записи макроса указатель мыши уже находился в ячейке `A1`, это действие не отражено в макросе. Такое упущение служит одной из причин, по которой вам требуется модифицировать макросы, записанные с использованием макрорегистратора. Затем макрос перемещает указатель мыши в ячейку `B2` и присваивает ей значение `2`. Наконец, макрос перемещает указатель мыши в ячейку `C3` и вводит в нее формулу, суммирующую два числа. Этот макрос можно изменить, добавив ссылку на пропущенную ячейку `A1` и комментарий и удалив одну лишнюю инструкцию (рис. 3.6).

Использование процедур

Процедуры мы уже использовали главе 2. Подпрограммы типа `Sub` — самый простой способ компоновки программного кода, и это единственный метод, допускаемый диалоговым окном Макрос. Следовательно, процедура всегда является основной точкой входа в программу.

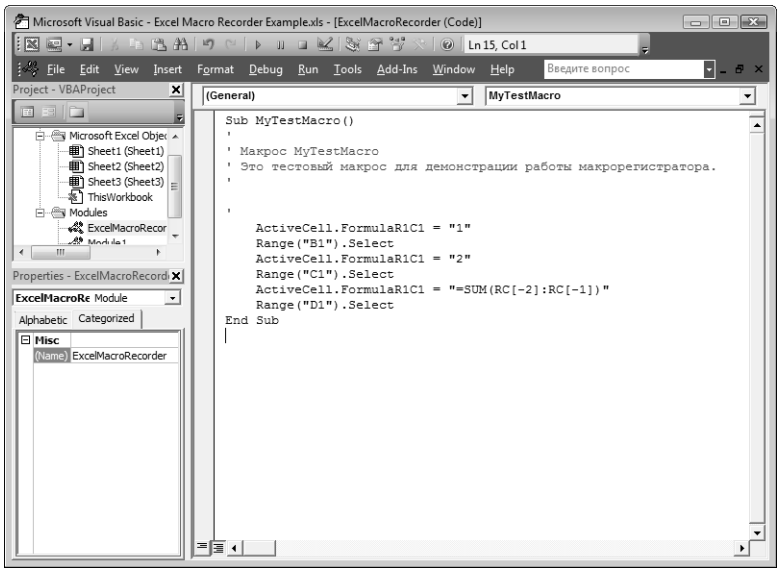

Рис. 3.5. Редактор Visual Basic используется для модификации макросов, созданных с помощью макрорегистратора

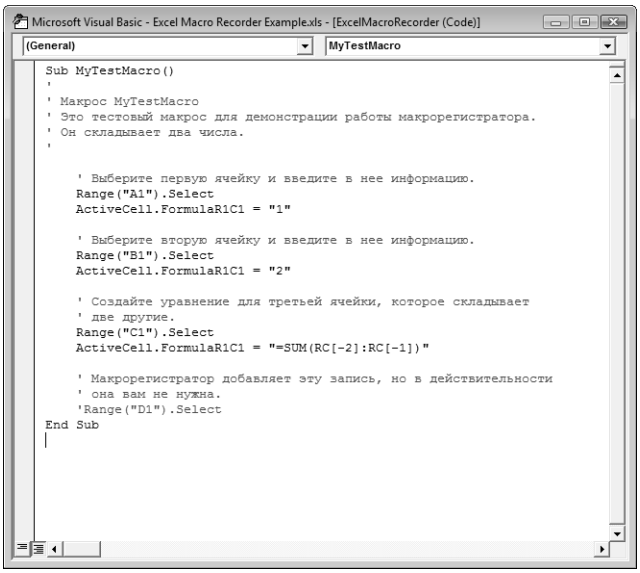

Рис. 3.6. Чтобы улучшить макрос, внесите измененения в результаты работы макрорегистратора

Процедуры используются тогда, когда нет необходимости возвращать значение. Они удобны для отображения информационных сообщений, как, например, в главе 2. Процедура может модифицировать данные, она лишь не может возвращать значения — это прерогатива функций. В то же время вы можете использовать аргументы для изменения значений либо задействовать глобальные переменные (об этом будет говориться далее).

Многие пользователи VBA применяют процедуры как средство разбиения кода на части. Вместо создания текста программы длиною в километры, использование нескольких конструкций типа `Sub` позволяет разделить программу на фрагменты размером со страницу. Такое структурирование значительно облегчает чтение кода.

Использование функций

Поработав некоторое время с процедурами, вам может показаться, будто функции вовсе не нужны. Однако подпрограммы типа `Function` используются для совсем других целей — когда требуется выполнить вычисления и вернуть результат.

Подпрограмма типа `Function` всегда возвращает значение, что отличает ее от подпрограммы типа `Sub`. По этой причине в функции включают программный код, который потребуется многократно вызывать в программе. Если вам, к примеру, потребуется обработать список имен, то можете создать функцию для обработки каждого имени в отдельности, а затем вызывать эту функцию снова и снова. Функция предоставляет обработанную информацию в виде возвращаемого значения. В главе 5 будет показано, как создавать повторяющийся программный код с использованием структур наподобие `Do...Until`.

Функции также используются для создания открытого кода, который не должен фигурировать в диалоговом окне Макрос. Это окно содержит только подпрограммы типа `Sub`.

Изменение параметров проекта

До сих пор мы работали с VBA, не настраивая многие параметры, и приводимые примеры базировались на тех настройках, которые обычно используются по умолчанию. Большинство программных уровней VBA предполагает определенную конфигурацию, включая уровень проекта. В этом разделе будут описаны различные параметры настройки проекта.

Чтобы открыть параметры проекта, щелкните правой кнопкой мыши на окне обозревателя проекта (окно Project редактора Visual Basic), а затем выберите пункт контекстного меню VBAProject Properties (Свойства VBA-проекта). Откроется диалоговое окно Project Properties (рис. 3.7). (Обратите внимание на то, что на рисунке заполнен ряд параметров, которые будут описаны далее.) Информация об окне Project обозревателя проекта приводилась в главе 1.

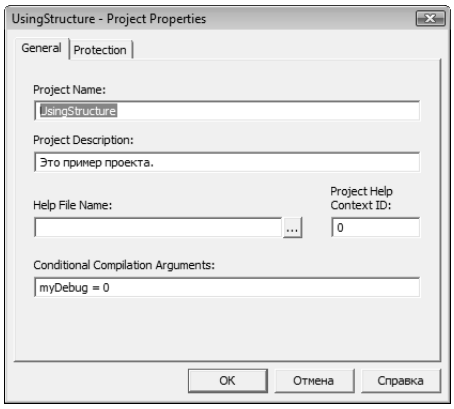

Рис. 3.7. Задайте базовую информацию о проекте

Описание проекта

Благодаря описанию становится легче следить за проектом при просмотре его в окнах редактора Visual Basic. Начните с присвоения проекту описательного имени. Оно не должно быть длинным, но и назвать проект просто `VBAProject` — не лучшая идея, поскольку это не дает представления о назначении проекта. Поле Project Name (Имя проекта), показанное на рис. 3.7, содержит имя, подходящее для данной главы, поскольку наши примеры являются демонстрацией того, как правильно структурировать программы.

В то время как значение поля Project Name отображается во многих местах, значение поля Project Description (Описание проекта) появляется только в окне Object Browser. В этом поле можно ввести пространное описание, чтобы облегчить отслеживание каждого компонента, отображаемого в окне Object Browser.

Некоторые пользователи не считают справочные файлы важной частью проекта, но в действительности это самый удобный способ описать, что делает ваш проект. Поле Help File Name (Имя справочного файла) указывает на то, где искать файл справки и какой файл использовать. Поле Project Help Context ID (Идентификатор контекстной справки проекта) содержит номер темы, относящейся к проекту. Обычно это порядковый номер ссылки на соответствующую страницу в оглавлении справки.

Условная компиляция

Условная компиляция позволяет создавать несколько копий программы, каждая из которых отличается своим способом выполнения. Обычно VBA проходит по списку инструкций и выполняет их последовательно одну за другой. Но при наличии инструкций условной компиляции можно выполнять одни действия на этапе тестирования и другие — когда разработка программы завершена. В главе 6 я покажу, как использовать условную компиляцию для отладки приложения. Здесь же для наглядности мы создадим очень простую программу. Прежде всего в поле Conditional Compilation Arguments (аргументы условной компиляции) введите ***myDebug = 0*** и щелкните на кнопке OK. Затем добавьте в модуль программу, текст которой приведен ниже (инструкции по созданию модуля приводились в главе 2).

```
Public Sub CheckConditional()
   #If myDebug = 0 Then
      MsgBox "В обычном режиме"
   #Else
      MsgBox "В режиме отладки"
   #End If
End Sub
```

Эта программа работает очень просто: если переменная `myDebug` установлена в `0`, программа должна отображать окно сообщения, с текстом `В обычном режиме`. Если же переменная установлена в любое другое значение, программа должна отображать окно с сообщением `В режиме отладки`. Запустите программу, и вы должны увидеть сообщение `В обычном режиме`.

Вновь откройте диалоговое окно Project Properties. Измените поле Conditional Compilation Arguments так, чтобы оно теперь содержало ***myDebug = 1***. Щелкните на кнопке OK, а затем запустите программу еще раз. На этот раз будет открыто окно с сообщением `В режиме отладки`.

Блокирование программного кода

В какой-то момент вы можете решить, что программный код необходимо “спрятать под замок”, дабы никто больше не смог его изменить. Это можно осуществить с помощью вкладки Protection (Защита) диалогового окна Project Properties (рис. 3.8). Просто установите

флажок Lock Project for Viewing (Запретить просмотр проекта), а затем дважды введите один и тот же пароль — сначала в поле Password (Пароль), а затем в поле Confirm Password (Подтвердить пароль). Для завершения процедуры щелкните на кнопке OK.

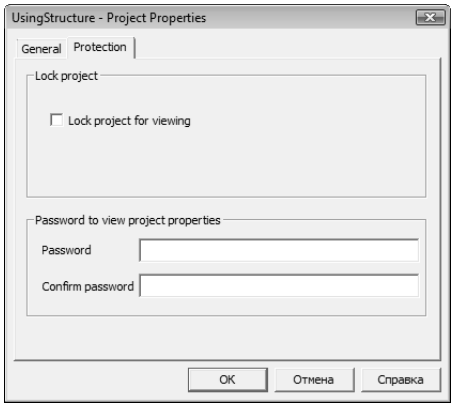

Рис. 3.8. Блокирование программного кода может уберечь его от любопытных глаз

Блокирование проекта может оказаться, в принципе, неплохой идеей, но по целому ряду причин следует тщательно взвесить, стоит ли это делать. Самая важная причина состоит в том, что будет невозможно разблокировать код без пароля. Если вы забудете пароль, проект останется заблокированным навсегда. Вторая причина заключается в том, что блокирование проекта не послужит препятствием для просмотра кода. Следовательно, если по каким-то соображениям необходимо скрыть код, VBA не очень подходит для решения этой задачи. (Для сокрытия кода вам понадобится полноценный компилятор наподобие Visual C++ или пакет типа Visual Studio Tools for Office, позволяющий интегрировать компилируемый код в приложения Microsoft Office.)

Не переусердствуйте при защите кода

Одна из опасностей, связанных с блокированием программного кода, заключается в том, что оно способно внушить обманчивое чувство защищенности. Хотя блокирование текста программы и не дает возможности пользователям-новичкам вносить изменения в текст, оно не гарантирует защиту от намеренного взлома. В Интернете можно найти десятки программ, разработчики которых предлагают разблокировать ваши документы Office, если вы забыли пароль. К сожалению, эти же программы могут сделать документы беззащитными перед взломщиками. Наиболее рациональный подход состоит в том, чтобы использовать блокирование кода для защиты пользователей-новичков от самих себя, а не в качестве средства защиты вашей интеллектуальной собственности.

Задание параметров компиляции

При первом запуске VBA делает определенные предположения относительно написанного вами программного кода. Эти предположения могут оказаться неточными, поэтому компания Microsoft позволяет изменить установки по умолчанию. Параметры, приведенные в табл. 3.1, помогут вам определить способ работы компилятора с программным кодом. (*Компилятор* считывает текст программы и транслирует слова в инструкции, понятные опе-

рационной системе.) Эти параметры следует размещать в самом начале модуля, модуля класса или формы, перед остальным кодом.

Параметр `Option Explicit` настолько важен, что его лучше всегда использовать его в программе. В листинге 3.1 представлен небольшой пример.

Таблица 3.1. Параметры компилятора VBA

Параметр	Описание
`Option Base <`*Цифра*`>`	Задает способ нумерации элементов массивов. Можно начать нумерацию с 0 или 1. В главе 9 я объясняю, как работать с массивами
`Option Explicit`	Опытные VBA-программисты всегда добавляют этот параметр. Он указывает, что вы намерены определять переменные перед их использованием. Это не только делает код более наглядным, но также помогает обнаруживать опечатки в коде. Если при стандартных настройках выражения `MyVar` и `MVar` будут восприняты как две разные переменные, то при наличии данного параметра VBA сообщит об ошибке
`Option Compare <`*Метод*`>`	Этот параметр позволяет устанавливать способ сравнения строк. При использовании побитового метода сравнения (`Binary`) VBA рассматривает *hello* и *Hello* как разные строки, поскольку регистр первой буквы различается. Использование метода посимвольного сравнения (`Text`) означает, что VBA рассматривает *hello* и *Hello* как одно и то же слово, так как регистр букв при это не учитывается. Метод сравнения баз данных (`Database`), используемый только в программе Access, учитывает порядок сортировки, принятый для базы данных
`Option Private Module`	Этот параметр позволяет сделать модуль закрытым, так что никакой другой модуль не сможет получить доступ к его содержимому. Понятия *открытый* (public) и *закрытый* (private) относятся к области видимости объекта. Об этом будет говориться далее

Листинг 3.1. Использование параметра *`Option Explicit`* для сокращения количества ошибок

```
' Заставить VBA проверять переменные.
Option Explicit

' Эта процедура завершится неудачей,
' поскольку переменная не определена.
Public Sub OptionCheck()
   MyVar = "Hello"
   MsgBox MyVar
End Sub

' Эта процедура завершится успешно.
Public Sub OptionCheck2()
   ' Определить переменную.
   Dim MyVar As String

   ' Присвоить переменной значение.
   MyVar = "Hello"

   ' Отобразить окно сообщения.
   MsgBox MyVar
End Sub
```

В обоих случаях процедура присваивает значение переменной `MyVar`. Процедура `OptionCheck` завершится неудачей, поскольку в ней переменная не объявляется перед своим использованием. VBA ничего не знает о переменной `MyVar`, поэтому не может ее использовать. Взгляните на вторую процедуру, `OptionCheck2`. Эта процедура работает успешно, поскольку в ней сначала объявляется переменная `MyVar`, а затем ей присваивает значение. Хотя использование параметра `Option Explicit` может показаться излишеством, в действительности это экономит время на "отлов" опечаток в программе.

Программирование: модульный подход

Не стоит чрезмерно усложнять программы, забывая о принципах абстракции. Самая большая ошибка, которую можно совершить, — написать одну длинную-предлинную программу, которую никто не сможет понять — даже вы сами. Программу, похожую скорее на "Войну и мир", чем на набор инструкций, обычно называют *спагетти-кодом*. Суть *модульного подхода* в программировании состоит в том, что программист разделяет программу на удобные модули и пишет по одному модулю за раз. Это позволяет писать легко модифицируемый программный код, понятный большинству людей. Модульный подход также дает возможность использовать готовые фрагменты кода при написании других программ.

Разработка плана приложения

В главе 2 было описано, как применять псевдокод при написании программ. Программа в той главе использовала только один модуль, поскольку была довольно простой. Однако на практике далеко не всякая создаваемая программа настолько проста. Иногда программа выполняет сложную задачу, и необходимо сначала создать *план приложения*. Если вспомнить игру Lego, то она состоит из отдельных блоков, но вы не увидите полной картины, пока не соедините блоки вместе. План приложения как раз и призван показать, как соединить блоки вместе, чтобы получить конкретный результат. Вы можете использовать блоки трех размеров:

- ✓ проекты;
- ✓ модули, формы и модули классов;
- ✓ процедуры и функции.

Представьте себе следующую метафору: у вас есть груда блоков (например, как в игре Lego) и вы мысленно видите картину того, что хотите сделать. Чтобы претворить замысел в жизнь, вы выбираете блоки, которые кажутся подходящими. Стандартные блоки можно выбрать из этой книги или справочных файлов VBA либо найти в Интернете, например на сайте FreeVBCode.com (`www.freevbcode.com`).

Для создания плана приложения можно применить несколько методик. Я обычно начинаю со списка основных задач, таких как печать отчета или поиск слова. Это не псевдокод. Вы не описываете процедуру, которую должен выполнить VBA. Вы думаете только о главных задачах. Может быть, программа решает всего одну основную задачу. В этом случае можно обойтись одним проектом, модулем или подпрограммой, как в примере из главы 2.

Модульное программирование и игра в Lego

Лично я называю метод, описываемый в этой главе, *Lego-подходом*, поскольку, по-моему, большинство людей знакомы с Lego и легко поймут аналогию с модульным методом программирования. В дальнейшем я иногда буду использовать метафору с игрой Lego. Только не следует путать все эти термины с объектно-ориентированным программированием, которое будет рассматриваться в главе 8.

Определение проекта

Возможно, вам никогда не понадобится определять более одного проекта для программы. Это особенно справедливо в отношении проектов Access и Excel, в которых все, что связано с данными, в большинстве случаев хранится в одном файле. В случае Word можно поместить все в один шаблон, если это единственный необходимый вам шаблон и он будет использоваться многими документами. Впрочем, при внимательном изучении можно обнаружить кое-что интересное касательно проектов. На рис. 3.9 показано типичное окно обозревателя проектов — Project. Обратите внимание на то, что в нем указаны три проекта: Normal, Project (Document1) и TemplateProject (MyLetter).

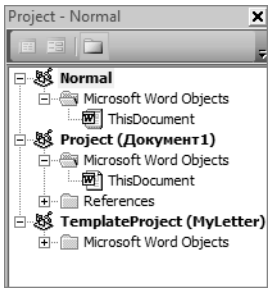

Рис. 3.9. Одна программа может охватывать несколько проектов

Программа Word всегда загружает шаблон `Normal`. Этот шаблон может иметь разные расширения, включая `DOT`, `DOTX` и `DOTM` в Office 2007. Более старые версии Word всегда используют расширение `DOT`. Word 2007 также понимает новый XML-формат (`DOTX`) и формат с разрешенными макросами `DOTM`. Если вы намерены написать макрос для Word 2007, с точки зрения совместимости лучше использовать формат `DOT`, но гораздо большая гибкость обеспечивается при использовании файлов типа `DOTM`. В файлах типа `DOTX` нельзя хранить макросы.

Можно также создать собственный шаблон. Пользовательский шаблон содержит специальные стили и макросы для конкретного типа документов, например писем. Аналогично шаблону `Normal`, пользовательский шаблон может иметь расширения `DOT`, `DOTX` и `DOTM`.

Наконец, в вашем распоряжении имеется сам документ, который тоже считается проектом. Документ может иметь расширения `DOC`, `DOCX` или `DOCM`. Макросы допускается хранить только в файлах формата `DOC` и `DOCM`. В данном случае я написал программу для отдельного документа, использующую специальные процедуры и функции из шаблона `Letter`, а также ряд стандартных подпрограмм из шаблона `Normal`. С этим примером можно будет ознакомиться в главе 13.

Еще одна ситуация, при которой необходимо использовать несколько проектов, возникает, когда пишутся программы, обеспечивающие взаимодействие нескольких приложений. Например, у меня есть программа для Word, запрашивающая импорт рисунка из CorelDRAW и преобразование его в формат, понятный для Word. Word получает преобразованное изображение и помещает его в текущий документ. Затем Word просит Access отыскать изображение в базе данных. Access выполняет это задание и возвращает описание рисунка в Word, после чего Word форматирует описание и помещает его над рисунком в текущем документе. Если бы мне приходилось выполнять весь этот цикл всякий раз, когда мне требовался рису-

нок, это занимало бы около 20 минут. Моя же программа может все сделать меньше чем за минуту, с неизменно отличным результатом.

Определить программу — значит понять, сколько строительных блоков требуется для выполнения задачи. Можно придерживаться простейшего подхода и использовать только один проект для нескольких задач. Но не стоит усложнять себе жизнь. Если необходимо использовать несколько проектов, VBA без труда обеспечит такую возможность.

Добавление модуля

Поначалу большинство ваших программ будет включать единственный модуль, в то же время любой проект начинается с определения модуля, пусть и единственного. Можно создать один огромный модуль, содержащий все когда-либо написанные вами программы, но это будет просто каша. Гораздо лучше создать один модуль для дисковых утилит, а другой — для утилит обработки данных. Ищите вариант, который окажется наиболее удобным.

По мере усложнения программ возникнет потребность организовать их взаимодействие с пользователем. Эта задача требует добавления форм в программу. Старайтесь создавать формы, которые запрашивают однотипную информацию, — не сбивайте пользователя с толку, перегружая формы ненужными элементами. (Описание форм и способов их создания можно найти в главе 7.)

Работа с объектами подразумевает использование классов. Чтобы создать специальный объект, включите в приложение модуль класса. Как и при добавлении форм, необходимо добавить отдельный модуль класса для каждого нового объекта. При этом нужно обеспечить, чтобы каждый создаваемый объект выполнял только одну задачу. (Методы работы с объектами описаны в главе 8.)

Проектирование процедур

В основе большинства примеров программ в этой книге лежит один проект и один модуль. Скорее всего, в большинстве ваших будущих программ будет использован такой же подход. В таком случае стоит подумать над тем, как разделить программу на процедуры и функции, о которых рассказывалось ранее.

При работе на уровне процедур и функций следует оформлять программный код в виде логических фрагментов. Каждая подпрограмма должна решать конкретную задачу. Вновь обратимся к метафоре Lego. Каждый игровой блок — это отдельный модуль. Требуется писать процедуры и функции так, чтобы можно было легко отделить их друг от друга. Примеры, приведенные в этой книге, демонстрируют различные способы разделения программного кода на части, поскольку в программировании это один из важнейших навыков.

Несложно понять, как разделить на части простую программу. Хорошим примером может служить так часто используемая нами функция `MsgBox`. Посмотрите, как она работает: вы передаете ей некоторую информацию, а она берет на себя заботу относительно вывода на экран окна сообщения. Вам не нужно беспокоиться о том, как функция `MsgBox` справляется с этой задачей — достаточно лишь вызвать ее.

Написание инструкций

После завершения подготовительной работы у вас имеется один или несколько проектов, содержащих один или несколько модулей с как минимум одной подпрограммой. Все блоки соединены вместе, но сами они пусты. В конце концов, программу образуют инструкции, написанные вами. Предварительная организационная работа упрощает задачу программирования инструкций, поскольку позволяет далее сконцентрироваться на решении отдельных задач.

В главе 2 я подчеркивал важность упорядоченного подхода к написанию инструкций. Следует начинать с написания псевдокода, а затем преобразовать его в инструкции, понятные VBA. Тем самым одновременно создается и программный код, и документация к нему. Однако не следует забывать еще кое о чем. Как будет показано далее, важно грамотно использовать пробелы и пустые строки, чтобы сделать код максимально наглядным. Необходимо также вставлять комментарии, поясняющие работу программы.

Создаем первую подпрограмму

Большинство приложений Microsoft Office располагает диалоговым окном Свойства (рис. 3.10), которое содержит вкладку Документ с перечнем свойств документа. На этой вкладке можно узнать основные характеристики документа, такие как имя автора и название компании, где был разработан документ. На других вкладках приводятся дополнительные статистические данные, например количество слов, содержащихся в документе. Дополнительную информацию можно получить, обратившись к теме BuiltinDocumentProperties (Встроенные свойства документа) в справочной системе VBA.

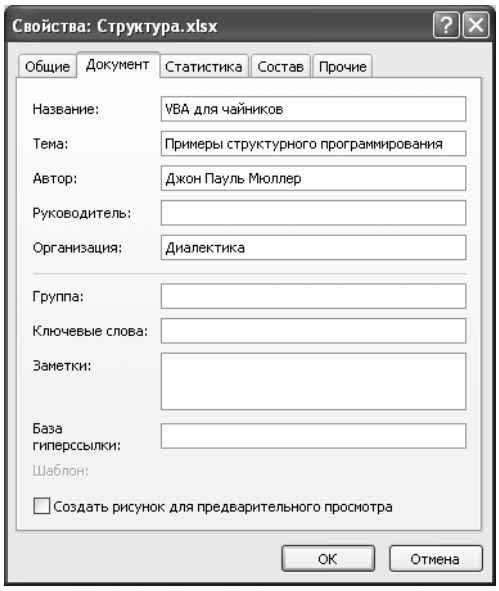

Рис. 3.10. На вкладке Документ перечислены основные свойства документа

В приложениях Office 2007, основанных на интерфейсе ленты, диалоговое окно Свойства вызывается не так, как раньше. Ниже описывается, как это сделать.

1. **Щелкните на кнопке Office, чтобы открыть одноименное меню.**
2. **Выберите команду Подготовить⇨Свойства.**

   Отобразится стандартный список свойств.
3. **Щелкните на кнопке Свойства документа.**

В раскрывающемся списке отображается опция Дополнительные свойства.

4. **Щелкните на пункте Дополнительные свойства.**

Откроется диалоговое окно, показанное на рис. 3.10.

Это первый пример, в котором мы непосредственно работаем с объектом. Свойство, которое мы будем использовать, называется `BuiltinDocumentProperties`. Оно доступно почти во всех приложениях Microsoft Office, но в каждом из них оно связано с разными объектами. При работе с Word вы обнаружите его у объектов `Word.Document` и `Word.Template`. Пользователи Excel найдут его у объекта `Excel.Worksheet`. Чтобы отыскать это свойство в программе, используйте обозреватель объектов (чтобы отобразить окно Object Browser, нажмите клавишу <F2>), который рассматривался в главе 1. Введите ***`BuiltinDocumentProperties`*** в текстовое поле Search, а затем щелкните на кнопке Search. На рис. 3.11 показан типичный результат выполнения описанных действий в Excel.

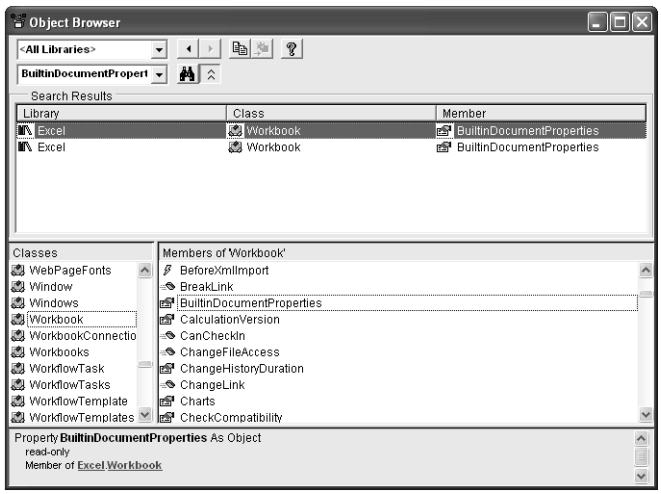

Рис. 3.11. В документации часто рассказывается об интересном свойстве, но не говорится, где его найти

Обратите внимание на текст внизу рис. 3.11. Здесь указано полное имя объекта, содержащего свойство. Мы воспользуемся им при написании программы, текст которой приведен ниже (листинг 3.2). При работе с другим приложением можете заменить объект `ActiveWorkbook` (Активная рабочая книга) другим объектом, например `Document` или `Template` в Word.

Листинг 3.2. Получение информации об авторе документа

```
Public Sub GetSummary()
   ' Объявление объекта DocumentProperty для хранения
   ' информации.
   Dim MyProperty As DocumentProperty

   ' Записываем в объект DocumentProperty информацию
   ' об авторе.
   Set MyProperty = _
      ActiveWorkbook.BuiltinDocumentProperties("Author")

   ' Отображение окна сообщения, содержащего значение свойства.
```

```
   MsgBox MyProperty.Value, vbOKOnly, "Author Name"
End Sub
```

Этот пример начинается с объявления переменной `MyProperty`. Она отличается от других переменных тем, что в действительности это объект `DocumentProperty`, который может хранить любое свойство документа, включая имя автора или название компании.

Следующая строка программы заносит в переменную `MyProperty` информацию об авторе, которая предоставляется объектом `BuiltinDocumentProperties("Author")`. Один объект можно сделать равным другому, если они принадлежат к одному типу объектов.

Если строка программы слишком длинная, в VBA можно продолжить ее на следующую строку, добавив символ подчеркивания (_) так, как показано в рассматриваемом примере. Символ подчеркивания служит в качестве *символа продолжения*. Его следует использовать, чтобы строки программы помещались на экране. Тогда текст программы будет легче читать.

Последняя строка программы отображает свойство `Value` объекта `MyProperty`. Именно так VBA работает с объектами. В данном случае вы запрашиваете значение одной из переменных, хранящихся в объекте `MyProperty`, и отображаете его в окне сообщения.

Запустите программу, чтобы увидеть диалоговое окно, содержащее имя автора документа. Чаще всего автором документа являетесь вы сами. Попробуйте изменить значение поля Автор, показанного на рис. 3.10, на какое-нибудь другое. Результат, выдаваемый программой, также изменится.

Создаем первую функцию

Пример из предыдущего раздела неплох (см. листинг 3.2), но он выдает лишь один фрагмент информации. Обычно пользователи VBA для выполнения повторяющихся задач прибегают к помощи функций. Соответствующий пример будет рассмотрен ниже.

В листинге 3.3 создается подпрограмма `GetSummary2`, которая многократно вызывает функцию `GetDocProperty`. При каждом вызове результат запоминается в специальной переменной. В конце подпрограмма отображает всю накопленную информацию.

Листинг 3.3. Использование функции для получения информации о документе

```
Public Sub GetSummary2()
   ' Объявление строки для хранения информации.
   Dim DocumentData As String

   ' Информационная метка.
   DocumentData = "Автор: "

   ' Получаем имя автора.
   DocumentData = DocumentData + GetDocProperty("Author")

   ' Добавляем дополнительную строку.
   DocumentData = DocumentData + vbCrLf

   ' Информационная метка.
   DocumentData = DocumentData + "Организация: "

   ' Получаем название организации.
   DocumentData = DocumentData + GetDocProperty("Company")
```

```
    ' Отображаем окно сообщения, содержащее значения свойств.
    MsgBox DocumentData, vbOKOnly, "Сводка"
End Sub

Private Function GetDocProperty(Name As String) As String
    ' Объявление объекта DocumentProperty для хранения информации.
    Dim MyProperty As DocumentProperty

    ' Заносим в объект DocumentProperty информацию об авторе.
    Set MyProperty = ActiveWorkbook.BuiltinDocumentProperties(Name)

    ' Возвращаем результат.
    GetDocProperty = MyProperty.Value
End Function
```

Листинг 3.3 начинается с процедуры `GetSummary2`. Многое в ней вам знакомо по предыдущему примеру. Но обратите внимание на то, как программа работает с переменной `DocumentData`. Программа формирует результирующий текст, добавляя новое содержимое в конец строки. Поэтому сначала переменная `DocumentData` содержит значение `"Автор: "`, а затем к нему добавляется имя автора с помощью функции `GetDocPropert`.

Еще одно новшество в этом примере — использование константы. Константа `vbCrLf` включает специальные символы, которые действуют так же, как нажатие клавиши <Enter> в конце строки текста.

Функция `GetDocProperty` представляет несколько новых идей. Первая из них — это возвращаемое значение. Функция может возвращать вызывающей программе значение. Вторая идея заключается в использовании аргумента. *Аргумент* — это входные данные для процедуры (`Sub`) и функции (`Function`). В данном случае через аргумент `Name` передаются данные для функции `GetDocProperty`.

Фактически текст функции `GetDocProperty` выглядит так же, как и в примере из предыдущего раздела. Однако в данном случае программа использует переменную `Name` в качестве аргумента свойства `ActiveWorkbook.BuiltinDocumentProperties`, а не константу. Также обратите внимание на то, что программа присваивает функции значение свойства `MyProperty.Value`. Это способ возвращения значения вызывающей программе.

Область видимости

Концепция области видимости не столь сложна, как кажется. В действительности *область видимости* просто определяет границу, до которой подпрограмма способна "видеть" переменные и до которой она раскрывает свои переменные другим подпрограммам. Например, при рассмотрении функции `MsgBox` нас интересует, как передать данные на ее вход и получить результат на выходе. Входные данные и выходной результат — это открытые (или видимые) элементы функции `MsgBox`. Для нас не столь важно то, что происходит внутри функции `MsgBox`. Эта внутренняя работа функции — та, которая остается "за кадром", — реализуется закрытыми (или невидимыми) переменными функции `MsgBox`.

Для вас, как для пользователя VBA, область видимости важна по двум причинам. Во-первых, если каждая часть программы будет видима для всех остальных, возникнет хаос, поскольку придется отслеживать слишком большой объем информации. Во-вторых, программы должны защищать свои данные от несанкционированной модификации. Короче

говоря, некоторые части программы необходимо открыть для использования, а другие — оставить невидимыми и, следовательно, защищенными.

Назначение области видимости

Как вы могли заметить, во всех примерах, которые приводились до сих пор применительно к процедурам (`Sub`) и функциям (`Function`), использовались два ключевых слова: `Public` и `Private`. Они могут быть связаны и с другими программными элементами. В частности, их можно использовать для определения области видимости переменных и классов. Область видимости оказывает влияние почти на все типы программных элементов, которые используются в этой книге, поэтому важно понять, как работают атрибуты `Public` и `Private`.

- ✓ ***`Public`* (Открытый).** Указывает на то, что помеченный элемент должен быть доступен другим программным элементам.
- ✓ ***`Private`* (Закрытый).** Указывает на то, что помеченный элемент должен быть скрыт от других программных элементов.

Делим сферы влияния

Лучше всего разобраться с областью видимости на конкретных примерах. Поэкспериментируйте с простейшими подпрограммами, чтобы понять, как изменение области видимости сказывается на их работе. Самое важное правило заключается в том, что область видимости влияет на работу элементов, расположенных вне текущего блока. Здесь в полной мере срабатывает модульный подход, описанный ранее.

Если модуль определен как закрытый с помощью параметра `Option Private Module`, значит, все, что находится внутри него, оказывается недоступным для внешних модулей. Даже если модуль содержит открытую подпрограмму, помеченную как `Public Sub`, только другие подпрограммы внутри модуля могут обратиться к ней — от всех подпрограмм вне модуля она скрыта.

Листинг 3.4 демонстрирует некоторые принципы области видимости.

Листинг 3.4. Использование глобальных переменных

```
' Объявляем закрытую глобальную переменную.
Private MyGlobalVariable As String

Public Sub GlobalTest()
   ' Присваиваем значение глобальной переменной.
   MyGlobalVariable = "Hello"

   ' Отображаем значение.
   MsgBox MyGlobalVariable

   ' Вызываем подпрограмму GlobalTest2.
   GlobalTest2

   ' Отображаем значение, возвращаемое в результате вызова.
   MsgBox MyGlobalVariable
End Sub

Private Sub GlobalTest2()
   ' Демонстрируем, что глобальная переменная
   ' действительно доступна.
   MsgBox MyGlobalVariable
```

```
    ' Изменяем значение глобальной переменной.
    MyGlobalVariable = "Goodbye"
End Sub
```

Обратите внимание на то, что переменная `MyGlobalVariable` — закрытая. Вы не можете получить доступ к ней вне текущего модуля. Однако обе подпрограммы в этом модуле свободно обращаются к ней.

В программе есть еще один нюанс: процедура `GlobalTest` — открытая, а `GlobalTest2` — закрытая. Чтобы проверить это, откройте диалоговое окно Макрос с помощью команды Сервис⇨Макрос⇨Макросы. В списке вы увидите подпрограмму `GlobalTest`, тогда как подпрограмма `GlobalTest2` в нем отсутствует.

Наберите в редакторе Visual Basic пример программы, приведенный в листинге 3.4, и запустите ее, чтобы посмотреть, как элементы программы влияют друг на друга. Вы должны увидеть три диалоговых окна. Первое из них выводит сообщение `"Hello"`, поскольку подпрограмма `GlobalTest` заносит этот текст в переменную `MyGlobalVariable`. Второе диалоговое окно тоже выводит сообщение `"Hello"`, поскольку переменная `MyGlobalVariable` доступна и в подпрограмме `GlobalTest2`, даже несмотря на то что значение этой переменной было изменено подпрограммой `GlobalTest`. Наконец, третье диалоговое окно выводит сообщение `"Goodbye"`, потому что подпрограмма `GlobalTest2` присвоила переменной `MyGlobalVariable` другое значение.

Удобство чтения программы

Вы, наверное, обратили внимание на то, как в листингах этой главы используются пробелы и пустые строки, чтобы придать тексту программы более наглядный вид. Если вводить инструкции непосредственно одну за другой, это не повлияет на работу программы. VBA не интересует пустое пространство: пробелы, символы табуляции и пустые строки. Однако об этом должны позаботиться вы, поскольку текст программы без всего перечисленного становится практически нечитаемым.

Заметьте, что за каждой парой “комментарий — инструкция” следует пустая строка. Она говорит тому, кто читает текст программы, что он достиг конца текущей инструкции или подпрограммы. Другой важный элемент оформления листингов — это отступы. В примерах данной главы отступы используются внутри текста процедур и функций, чтобы нагляднее выделить тело подпрограммы.

Документирование программного кода

Метод псевдокода, описанный в главе 2, представляет собой хороший способ начать документирование программного кода. Однако в определенный момент может потребоваться добавить дополнительную информацию, если сведений в виде псевдокода окажется недостаточно, чтобы разобраться в работе подпрограммы. В программу часто добавляются и другие комментарии, такие как имя разработчика, название проекта и т.п.

Основные комментарии

Комментарии могут принимать различную форму. Комментарии в виде псевдокода пишут почти все программисты, поскольку это наиболее естественный вид комментариев. Разработчики быстро переходят к добавлению документирующих описаний, таких как сведения о том, кто написал программу или когда она была впервые написана, наряду со списком изменений, внесенных в код.

Важно подробно комментировать, почему вы выбрали именно такой способ написания программы. Просто сказать, что программа выполняет определенную задачу, недостаточно, поскольку, как правило, одну и ту же задачу можно реализовать несколькими способами. Объяснение того, на чем основывается ваш выбор, позволит уменьшить количество ошибок в процессе последующей доработки программы, когда ваши навыки программирования улучшатся.

Как опытный программист, вы также должны включить в комментарии описание сделанных вами ошибок, если полагаете, что кто-то другой может совершить такие же. Подобные комментарии часто помогали мне, поскольку, как правило, я заносил их в свои заметки. Начиная новый проект, я заглядываю в свои записи, чтобы понять, чего следует избегать.

Почему необходимы комментарии

Используйте комментарии всегда и везде, если считаете, что они необходимы. Написание хороших комментариев может требовать значительных затрат времени и оказаться нелегким делом, поскольку они заставляют вас вновь и вновь думать о программе. В то же время плохо прокомментированные программы обычно становятся источником головной боли в процессе последующей их модификации. Я был свидетелем нескольких случаев, когда отсутствие комментариев заставляло компанию принять решение переписать программу с самого начала, вместо того чтобы оплатить услуги программиста, который смог бы разобраться в старом программном коде.

Что такое хороший комментарий

Хороший комментарий — тот, который вы можете понять. Не используйте необычных терминов — пишите простым языком, который всем понятен. Если чувствуете, что требуется объяснить то или иное решение, делайте это без колебаний. Хорошие комментарии должны отвечать на шесть вопросов: кто, что, где, когда, почему и как. Позаботьтесь о том, чтобы комментарии были исчерпывающими и полностью отвечали на вопросы, которые могут возникнуть у того, кто будет читать ваш программный код.