Глава 54

Объектно-ориентированное программирование в VBA

В этой главе...

- Знакомство с объектами
- Использование объектов в коде
- Использование методов
- Создание классов объектов

VBA является инструментом для разработки программного обеспечения на основе объектов. Понимание сущности объектов — ключ к программированию в VBA, особенно при создании диалоговых окон или передачи возможностей ведущей прикладной системы под управление пользователя.

Итак, что такое объект?

Сформулировать формальное определение объекта можно, но гораздо проще понять, что представляют собой объекты, используя примеры или рассматривая их функции.

Объекты могут быть компонентами приложения VBA

Простейший способ понять суть объектов — рассматривать их как часть приложения VBA и документов. Ячейка рабочего листа Excel является объектом, так же как и именованный диапазон ячеек. То же касается отдельных рабочих листов и рабочих книг, в которые входят ячейки, диапазоны и листы. Во всех офисных приложениях VBA панели управления и меню (так же, как кнопки и элементы меню) являются объектами.

Очевидно, что у объектов VBA существует определенная иерархия, в которой объекты одного типа содержат объекты других типов. Иерархия объектов рассматривается в разделе "Понятие о модели объекта" далее в этой главе. В настоящий момент, однако, хотелось бы сосредоточиться на отдельных объектах.

Осмысление объектов может оказаться непростым делом

Конечно, трудно потрогать нарисованную фигуру, ячейку рабочего листа или кнопку панели управления, но можно думать о них как о конкретных вещах. Можно, по крайней мере, мысленно вырезать круг и перенести его на другой лист бумаги. Можно вписать числа в ячейки рабочего листа или нажать на кнопку. Помимо конкретных элементов, приложения VBA предлагают большой набор абстрактных объектов. Вот некоторые примеры.

- В Word существует объект `Style`, который представляет собой комбинацию характеристик форматирования абзаца.
- Объект Word `FileSearch` "представляет функциональность диалогового окна Открыть", цитируя соответствующий раздел справки. Отметим, что этот объект представляет не диалоговое окно, а именно его функциональность.
- В Excel есть объект `CustomView`, который работает с представлениями (в Excel представления определяют вид рабочей книги и ее настройки при печати).
- VBA также имеет несколько объектов, которые можно использовать в приложениях VBA. Например, объект `Collection` — это набор переменных и других объектов, с которыми можно работать как с отдельным элементом, невзирая на содержащиеся в нем типы данных.

Практическое определение

Обычно тяжело представить объект VBA как нечто материальное. И это к лучшему: чем дальше удастся отойти от подобных представлений, тем проще работать со всем многообразием предоставляемых объектов. Определение, используемое программистами, на самом деле, достаточно простое. *Объект* — это именованный элемент, обладающий:

- **свойствами** (настройки, которые может проверять и изменять пользователь);
- **методами** (действия, которые может выполнять объект по запросу программы);

а также в некоторых случаях

- **событиями** (то, что происходит с объектом и на что он может отреагировать, автоматически предпринимая заранее заданные действия).

Вам может показаться, что термин *объекты* не подходит таким “богато одаренным созданиям”. Действительно, объекты больше похожи на живые существа, чем на мертвые глыбы. Тигр или кит обладает такими характерными чертами, как глаза, конечности и хвост — объект обладает *свойствами*. Лошадь или собака могут выполнять команды или убегать от опасности — объект имеет *методы* и *события*.

Все еще хочется услышать техническое определение? Попробуем следующее: *объект* — это именованный элемент программы, который содержит информацию и код, обрабатывающий эту информацию. Объект, говорят, *инкапсулирует* информацию и соответствующий код.

Классы объектов и отдельные объекты

Есть также другой аспект, о котором необходимо помнить: существует различие между определенным объектом и шаблоном, на котором основан объект. Отдельно взятый объект представляет один определенный документ, форму, ячейку рабочего листа или другой организованный элемент информации. Объект документа, например, включает в себя текст всего документа.

Класс объектов, с другой стороны, можно сравнить с пакетом планов строительства. Можно построить множество домов, используя один комплект планов, но никто не может жить в самих планах. Аналогично, класс содержит типы информации, которые могут храниться в объекте, и определяет методы, свойства и события объекта. Основываясь на описании, пользователь создает *экземпляр* (instance) класса — объект, в котором и хранится какая-либо информация. Пользователь может создать или

реализовать (instantiate) столько объектов класса сколько ему нужно, причем каждый будет существовать отдельно, и в каждом будет содержаться различная информация.

Что такое объектная модель?

Как ясно из вышесказанного, объекты VBA состоят в определенной иерархии. Помимо того, что каждый из них обладает собственными свойствами, методами и событиями, объект, возглавляющий иерархию, служит *контейнером* для одного или нескольких типов. Эти объекты, в свою очередь, содержат другие объекты и т.д.

Для любого приложения VBA особенности иерархических отношений между объектами называются *объектной моделью* приложения. Объектная модель, часто представляемая графически, определяет, какой объект содержит другие объекты и какие именно. На рис. 54.1 показано подобное представление объектной модели.

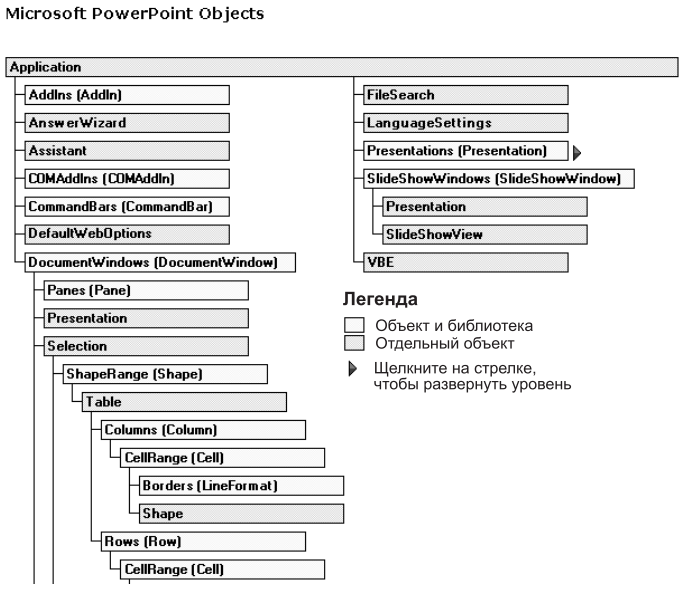


Рис. 54.1. Часть объектной модели PowerPoint

В самом верху иерархии объектной модели находится объект `Application`. Это контейнер для всех других объектов приложения, которыми может управлять пользователь. Любой пользовательский проект VBA также является объектом-контейнером. Он содержит все написанные пользователем модули кода, созданные формы и документ проекта (подробнее о проектах VBA см. главу 53).

Зачем нужна объектная модель

Поскольку VBA необходимо сообщать, с каким именно объектом выполняется работа, понимание объектной модели приложения VBA необходимо для успешной работы. По схеме объектной модели (типа той, что изображена на рис. 54.1) можно быстро определить ветвь, к которой принадлежит нужный объект. Из данного примера видно, что объект `Selection` содержится в объекте `DocumentWindow` (который является элементом библиотеки `DocumentWindows`). Далее будет описано, как использовать отношения между объектами объектной модели для определения в коде отдельных объектов, с которым будет работать программа.

Расширение объектной модели

В программах, написанных при помощи VBA, пользователь не ограничен в применении объектов одного приложения. Фактически, пользователь даже не ограничен приложениями VBA. Другие приложения и специализированные "компоненты" тоже подходят, если они соответствуют стандарту Microsoft Component Object Model (COM).

COM — это технические требования, описывающие, как объекты определяются в приложениях и других программных элементах и как они могут использоваться в других приложениях. Слово *автоматизация* указывает на способность приложений, основанных на COM, быть управляемыми другой программой. Между прочим, COM не является технологией, присущей исключительно VBA или даже Visual Basic. Некоторые из инструментов разработки программного обеспечения, такие как C++, понимают COM и могут получать доступ к объектам COM.

В любом случае, это открывает фантастические возможности по созданию мощных, специальных приложений VBA. К тому же позволяет быстро (не путать с "легко") создать приложение, получающее информацию из документов Word, рабочих листов Excel и даже приложений VBA, не входящих в семейство Office (например Visio). Специальное пользовательское приложение может выводить всю эту информацию в собственных окнах, использующих для вывода возможности компонента приложений. Разработка специальных приложений такого уровня описана в главе 57.

Формы VBA также являются объектами

Форма — общий термин для любого специального окна или диалогового окна, разработанного при помощи VBA. Следует понимать, что формы VBA также являются объектами. Они представляют собой категории, которые содержат информацию, описывающую расположение формы и набор инструментов для работы с этой информацией. Принятый термин для обозначения формы — объект `UserForm`.

Подобным образом, элементы управления формы (каждая кнопка, флажок и т.п.) тоже являются объектами. VBA предлагает различные типы объектов для каждого элемента управления.

Формы тоже обладают свойствами, методами и событиями

Поскольку VBA — визуальный инструмент разработки, писать код для создания и размещения формы и элементов управления не нужно. Но, так как формы и элементы управления, в полной мере, являются объектами VBA, с ними можно работать как с любым другим объектом, используя

- **свойства** для изменения вида или поведения формы или элемента управления во время работы программы;
- **методы**, чтобы заставить форму или элемент управления выполнить какое-либо действие, например стать видимым или переместиться в новое место;

- **события**, чтобы сообщить форме или элементу управления как реагировать на такие действия пользователя, такие как нажатие клавиши, щелчок кнопки мыши, или на какое-либо событие.

Подход, который применяется при написании кода для доступа к конкретной форме с целью использования ее свойств или методов, абсолютно такой же, как и при работе с другими объектами. Применение его к формам и элементам управления подробно описано в главе 56.

Формы входят в объектную модель

Как и любой другой объект VBA, формы прекрасно вписываются в принцип объектной модели. Каждый объект `UserForm` может принадлежать одновременно двум библиотекам объектов: проекту VBA, в котором хранится форма, и библиотеке `UserForms`, которая хранит все формы загружаемые программой. Объект `UserForm` является контейнером для объектов библиотеки `Controls`, которая, в свою очередь, содержит все отдельные элементы управления, входящие в форму. Данные отношения проиллюстрированы в диаграмме на рис. 54.2.

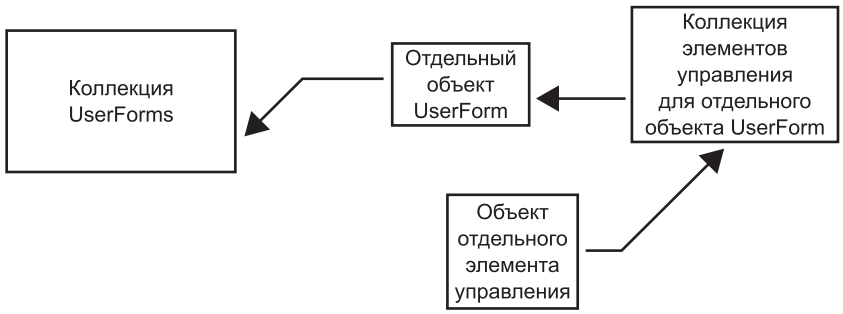


Рис. 54.2. На этой диаграмме отображено, как объекты элементов управления хранятся в библиотеках, которые, в свою очередь, принадлежат объекту `UserForm`, являющемуся частью библиотеки `UserForms` и т.д.

Использование объектов в коде

После того, как были заложены теоретические основы, на очереди практическое применение объектов VBA. Хотя концепция объектов может оказаться сложной для понимания, использовать объекты довольно легко. Поскольку объекты имеют имена, всегда известно, с чем выполняется работа. Чтобы изменить свойство объекта или его методы, в коде достаточно ввести имя объекта, точку и затем название свойства или метода. Например, `MyWorksheet.Calculate` определяет метод `Calculate` для объекта рабочего листа Excel, называемого `MyWorksheet`.

Может возникнуть вопрос, "Откуда известно, что за имя объекта стоит вначале?". Хороший вопрос. Достаточно понять, что объект можно вызвать по имени.

Какой объект выбрать

Любое приложение VBA имеет собственную объектную модель (иерархически структурированный набор объектов), и каждый из объектов имеет собственный набор

свойств, методов и событий. Внимательно ознакомьтесь с особенностями объектной модели приложения, чтобы использовать ее объекты в программах.

Файл справки VBA для каждого приложения Office содержит раздел, посвященный карте объектной модели приложения (см. рис. 54.1). Эта графическая схема объектной модели приводится по умолчанию, но если при открытии справки ее нет, в системе поиска VBA выберите вкладку Contents (Содержание) и просмотрите разделы Microsoft Outlook Objects, Microsoft Excel Objects и т.д.

На схеме объектной модели голубым цветом представлены одиночные объекты; желтым — библиотеки объектов. Чтобы перейти к разделу справки, посвященному отдельному объекту или библиотеке, просто щелкните на соответствующем прямоугольнике схемы объектной модели. Как правило, в каждом разделе есть ссылки на другие разделы, описывающие свойства, методы и события.

Окно Object Browser (Объекты) редактора Visual Basic — еще один важный инструмент для просмотра отношений между различными объектами приложения и работы с их свойствами, методами и событиями (см. главу 52).

Получение и изменение свойств объекта

Свойства объекта — это именованная характеристика объекта. Она описывает какой-либо из аспектов вида, поведения, содержания или, если угодно, "происхождения" объекта. Объект `Document` должен иметь свойство `Pages`, которое содержит количество страниц документа. Объект `Shape` имеет свойство `Fill`, определяющее цвет формы. Объекты `CommandButton` (управляющие кнопки диалоговых окон) всегда имеют свойство `Caption`, которое содержит текст, отображаемый на кнопке.

Свойства, связанные с поведением, определяют способ, которым объект реагирует на различные вводные сигналы. Элементы управления в формах тоже являются объектами, и у них есть свои свойства, например `Enabled`, которое определяет, отвечает ли элемент управления на такие события, как щелчок мыши. Некоторые свойства могут принимать огромный диапазон значений. Другие ограничены списком заранее определенных величин, например `Mauve`, `Teal` или `Chartreuse`. Многие свойства могут принимать только одно из двух возможных значений: `True` или `False`, `Hot` или `Cold`, `Wet` или `Dry`. В любом случае, для возврата текущих значений определенного свойства и изменения их на нужные используйте простые операторы VBA.

Программа VBA может возвращать текущие значения, хранимые в большинстве свойств, что позволяет решить, предпринимать или нет какие-либо действия или просто вывести в окне текущие настройки. И наоборот, программа может изменять значение свойства с целью изменить вид или поведение объекта, но только если свойство позволяет (многие свойства могут быть возвращены, но не изменены).

Чего нельзя делать с некоторыми свойствами

Знать, как получить доступ к значению свойства, еще не значит получить его. Изначально, некоторые свойства позволяют получить значения, но не изменять их. Эти свойства называются свойствами *только для чтения*. В меньшем количестве присутствуют свойства *только для записи*, т.е. можно задать значение, но нельзя получить текущее. Однако большинство свойств представлено типом *чтение/запись*, т.е. такие свойства могут быть как записаны, так и считаны.

Установки свойства — тоже данные

Хотя функции свойства описывают характеристики объекта, следует понимать, что параметр, определяющий эту характеристику, состоит из определенных данных, которые не отличаются от данных, содержащихся в переменных VBA. Таким образом, можно представить свойство как более-менее постоянную переменную, которую не нужно объявлять.

При таком подходе становится понятным, что каждое свойство хранит определенный тип данных, совсем как переменные. Свойства, которые могут принимать только одно из альтернативных значений, имеют тип `Boolean`. Некоторые из свойств являются строками, некоторые целыми числами, некоторые числами с плавающей точкой и т.д. Свойства могут быть даже объектами (подробнее о типах данных, используемых в VBA, см. главу 53).

Получение текущего значения свойства

Чтобы узнать или получить текущее значение свойства, используйте свойство так, словно это функция или процедура `Function`. Т.е. достаточно в коде присвоить свойство переменной. Переменная должна быть того же типа, что и свойство.

В приведенном ниже примере объект представляет собой вопрос выпускного компьютерного теста в школе. Интересующее нас свойство отвечает за уровень сложности вопроса по десятибалльной шкале:

```
Dim intHowTough As Integer
intHowTough = objTestQuestion.DifficultyLevel
```

В первом операторе объявляется переменная для хранения текущего значения свойства; второй оператор присваивает переменной это свойство.

Зачем нужно текущее значение переменной? Часто в условных операторах оно используется для решения, предпринимать или нет какое-либо действие. (В этом случае подошло бы что-то вроде "Если `DifficultyLevel` вопроса больше 8 и ответ правильный, повысить балл ученика вдвое"). Также можно сохранить значение свойства в переменной, чтобы затем присвоить данное значение этому же свойству или другим похожим объектам.

Если значение свойства применяется только однажды, необязательно присваивать его переменной, можно прямо использовать его в выражении, например:

```
If objTestQuestion.DifficultyLevel > 8 Then
    intTestScore = intTestScore + (intPoints * 2)
End If
```

Такой способ удобен, но не забывайте, что программа существенно замедляется, если в ней постоянно происходит получение значения. Если значение используется больше одного-двух раз, лучше хранить его в переменной — VBA может обработать значение обычной переменной гораздо быстрее, чем значение свойства.

Изменение значения свойства

Следует помнить, что свойства — просто разновидность переменных, и им можно присваивать значения так же, как и переменным, помещая название свойства слева от знака равенства, а новое значение — справа. Оператор

```
objMetalTune.GrungeFactor = 999
```

устанавливает значение свойства `GrungeFactor` объекта `objMetalTune`, предположительно меру деформации, шумности и т.п., равным 999.

```
ObjMetalTune.Ballad = False
objMetalTune.Title = "У меня блохи, и это плохо."
```

Свойства по умолчанию

Многие объекты обладают свойствами по умолчанию. Получить или установить его значение можно, используя только объект, но не указывая свойства. Пользуясь последним примером, предположим, что свойством по умолчанию для объекта `objMetalTune` является `Title`. В таком случае можно упростить последний оператор:

```
objMetalTune = "У меня блохи, и это плохо."
```

Свойства по умолчанию удобны до тех пор, пока пользователь помнит, какое из них установлено по умолчанию. Если есть сомнения, лучше использовать название свойства.

Объекты в роли свойств

Как упоминалось ранее, свойство одного объекта может описывать другой объект, что позволяет работать в коде с подчиненными объектами, принадлежащими определенному контейнеру, так же, как и с другими свойствами контейнера. Например, в выражении

```
Workbook.ActiveWorksheet
```

`ActiveWorksheet` — свойство объекта `Toolbar`, но его значением является объект рабочего листа. Такое использование свойств объекта важно для определения объектов, с которыми нужно работать. Подробнее см. раздел "Определение объекта, нужного для работы" далее в этой главе.

Генеалогическое древо

Свойства объекта могут определить другие принадлежащие им объекты, а могут сообщить, к какому контейнеру объектов принадлежат они сами. В Excel, если есть объект `Chart`, который хранится в переменной, а нужно узнать, к какому документу он принадлежит, выражение

```
Chart.Parent
```

возвратит ссылку на нужный документ. Если необходимо знать, к какому приложению принадлежит объект, можно перейти в самый верх иерархической структуры, получив значения свойства `Application`:

```
Chart.Application
```

Методы

Метод — именованное действие, которое выполняется при вызове метода. Вообще, методы — это те же процедуры, привязанные к определенным объектам. Поскольку код каждого метода является частью объекта, объекту известно, что делать, когда вызывается метод.

Объекты формы Office, например, имеют метод `ScaleHeight` (для изменения вертикальных размеров окна) и метод `IncrementRotation` (определяет чередование окон). Объект ячейки рабочего листа имеет метод `Calculate` (пересчитывает значение ячейки) и метод `Clear` (очищает содержимое ячейки). Объект `Document` (представляющий весь документ) обычно имеют методы `Print` и `Save`.

Вызов метода

Чтобы вызвать метод, введите имя объекта, точку и затем имя метода. Предположим, что объект `objJazzTune` представляет цифровую джазовую композицию в мультимедийной программе. Допустим, у объекта есть метод `Play`. Вот как его можно вызвать:

```
objJazzTune.Play
```

Подход к вызову методов совпадает с тем, что применяется при вызове процедур и функций VBA (см. главу 53).

Как и в случае со свойствами, различные классы объектов могут иметь методы с одинаковыми именами. Объекты, содержащие группы элементов или другие объекты, как правило, имеют метод `Add`, подробно описанный далее в этой главе.

Изменение свойств при помощи методов

Хотя это и не удивительно, следует заметить, что метод может изменять значение одного или нескольких свойств. Например, объект `objJazzTune` может иметь свойство только для чтения `TimesPlayed`, которое можно изменить только при помощи метода `Play`, но его нельзя получить через операторы присваивания типа `intPlaybacks = objJazzTune.TimesPlayed`. Некоторые объекты даже имеют специальные методы, единственная задача которых — устанавливать значения свойств.

События

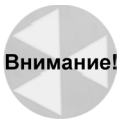
Событие — это что-то происходящее с объектом, вследствие чего он может выполнить какое-либо заранее определенное действие (на жаргоне VBA, когда говорят, что объект "имеет" события, подразумевают, что объект может обнаруживать и распознавать события). К событиям относятся:

- физические действия пользователя, такие как щелчок мыши, передвижение курсора или нажатие клавиши;
- то, что происходит с объектом под управлением программных средств; например открытие или закрытие документа, добавление или удаление страницы.

Приложение VBA определяет, какие события может распознавать определенный объект. При этом нужно написать код, задающий поведение объекта при выполнении события.

Наиболее часто пишется код для обработки событий, происходящих с формами и элементами управления форм (кнопками, текстовыми окнами и пр.). Щелчок пользователя на кнопке инициирует действие, только если написан код для обработки события `Click` этой кнопки. Программирование обработки событий для форм описано в главе 57.

Однако часто бывает нужно написать код обработки событий, происходящих с другими типами объектов. В Word события распознаются объектами `Application` и `Document`. Можно написать код, который будет автоматически выполняться при выполнении этих событий. Используемый подход изменяется в зависимости от приложения, а иногда и в границах одного приложения, поэтому ознакомьтесь с документацией приложения и файлами справки.



Следует уточнить: пользователь не вызывает событие из кода — объект автоматически предпринимает определенные действия, когда происходит событие.

Определение объекта, нужного для работы

Чтобы выполнить какие-либо действия с объектом, следует указать VBA, с каким объектом нужно работать. Для этого используется *объектное выражение* — специальное выражение VBA, которое уникальным образом определяет объект. На деле, значение, вычисляемое VBA и основанное на объектном выражении, называется *объектной ссылкой* — значение, которое сравнимо для объекта с номером дома.

Определение правильного объектного выражения — половина успеха. Далее можно облегчить выполнение задач, создав именованные переменные для работы с объектом, при помощи выражения для присвоения соответствующей объектной ссылки переменной. Затем можно ссылаться на объект в коде, используя имя переменной.

Объектное выражение — фрагмент кода, выражение, которое "указывает" на определенный объект. Используя соответствующее объектное выражение, можно изменить свойства объекта, выполнить его методы или присвоить объект переменной.

Изложенные выше идеи применяются для ежедневной работы с VBA, однако их не всегда легко понять с первого раза. Поскольку программа может работать с множеством объектов одного типа, полное объектное выражение должно определять все объекты, содержащиеся в нужном. Это все равно, что попросить позвать "мальчика". Сразу возникает вопрос — "Какого?". А если сказать, что нужно позвать самого старшего мальчика живущего в третьем доме по улице Ленина в городе Нежине Киевской области в Украине, ответного вопроса не возникнет. (Хотя, конечно, можно спросить "Зачем?")

Однако если человек уже находится в доме №3 по улице Ленина, и там живет только один мальчик, вполне целесообразна команда типа "Покорми мальчика". Так же и в VBA: если контекст определен, указывать весь список объектов не нужно.

Свойства могут быть объектами

Как упоминалось ранее, свойство объекта может быть другим объектом. Также говорилось о том, что среди объектов существует своя иерархия, в которой один объект, например, документ, служит контейнером для других, например, страниц или рабочих листов.

Связь между этими вещами очевидна: если объект содержит дочерние объекты, их можно определить через свойство первого объекта. Выражение, используемое для определения данного свойства, — объектное выражение. Например, следующее выражение определяет объект в документе Word:

```
ThisDocument.Section(2).Range
```

Это выражение содержит не одну, а две точки, т.е. `Range` является свойством второго объекта `Section`, который, в свою очередь, выступает свойством объекта `ThisDocument`.

Получение объектов

В диаграмме на рис. 54.3 изображены отношения между объектами, о которых шла речь в предыдущем объектном выражении.

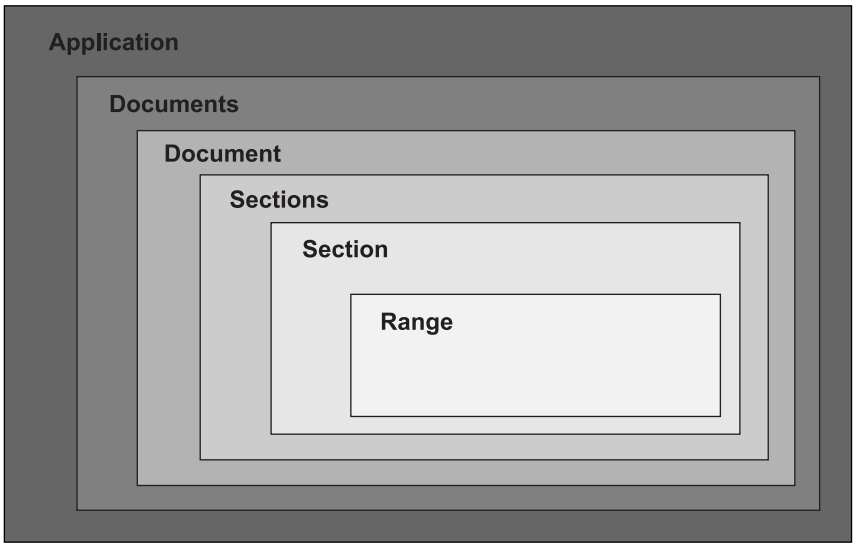


Рис. 54.3. Иерархические отношения между объектами

Объект `Application` находится в самом верху лестницы объектов в большинстве объектных моделей VBA. Однако, как правило, его не нужно включать в объектные выражения — VBA хватит ума понять, что пользователь работает с объектами текущего приложения, пока не определено обратное.

Объект `Application` содержит объект `Documents`, который является библиотекой, хранящей все открытые документы. Если существует необходимость работать с определенным документом, можно определить его как элемент объекта `Documents`. Например `Documents(5)` — пятый объект `Document` в библиотеке `Documents`. Выражение `ThisDocument.Sections(2).Range`, однако, начинается со специального ключевого слова `ThisDocument`. Во многих приложениях VBA `ThisDocument` служит заменой объекту `Document`, связанному с проектом.

В Word каждый объект `Document` обладает свойством `Sections`. Оно, в свою очередь, относится к библиотеке `Sections` — объекту, представляющему набор всех разделов документа. Т.е. первая часть предыдущего выражения, `ThisDocument.Sections`, определяет библиотеку `Sections`, принадлежащую объекту `ThisDocument`. При определении объекта `Sections` можно выбрать только один элемент библиотеки. Таким образом `Sections(2)` ссылается на второй раздел документа.

Что касается последней части выражения, то трюк заключается в следующем. Хотя `.Range` определяет свойство объекта `Section`, значение этого свойства — объект `Range`. Все выражение, таким образом, представляет собой ссылку на этот объект `Range`. Использование выражения, которое прокладывает себе путь к определенному объекту через подобную иерархию, на жаргоне VBA называется "получение объекта".

Создание объектных переменных

Никто не любит вводить длинные, сложные объектные выражения, подобных тем, что рассматривались в предыдущем разделе. Если программа использует один и тот же объект несколько раз, создайте переменную, хранящую ссылку на объект. Затем, вместо того, чтобы вводить целое объектное выражение, достаточно ввести имя переменной. Оно короче, запомнить и ввести его легче чем изначальное объектное выражение. У объектной переменной есть еще два преимущества. Прежде всего, она позволяет делать код быстрее. VBA может сразу определить объект, а не искать среди массы свойств и объектов. Таким образом можно создать гибкий код, который определяет, какой объект должен храниться в переменной во время выполнения программы.

Схематически техника создания объектной переменной включает два этапа:

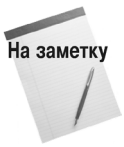
1. Объявление переменной, которая будет использоваться для хранения ссылки на объект.
2. Присвоение нужного объекта переменной.

Следующие два раздела подробно описывают каждый из этапов.

Объявление объектной переменной

Объектные переменные объявляются так же, как и переменные других типов. Стандартный метод подразумевает использование оператора `Dim` (подобно другим типам данных, хотя для объявления объектных переменных можно вместо `Dim` использовать ключевые слова `Public`, `Private` или `Static`). Вот два примера объявления объекта:

```
Dim objGreatBigObject As Object ' родовой объект
Dim objShapeObject As Shape ' объект типа Shape
```



Различие между двумя объявлениями очевидно. Первый оператор объявляет объектную переменную, но не определяет тип объекта, который она содержит. Данную переменную можно использовать для хранения различных типов объектов. Второй оператор объявляет определенный тип или класс объекта, который будет храниться в

переменной. VBA не позволит разместить в этой переменной объект другого типа.

Эти два типа объявлений относятся к *динамическому связыванию*, в котором не определяется тип класса, и *раннему связыванию*, в котором объявляется переменная определенного объекта класса. Старайтесь использовать раннее связывание, объявляя объектную переменную как определенный класс. Раннее связывание обладает следующими преимуществами.

- **Меньшее количество ошибок.** Как упоминалось ранее, VBA не позволит присвоить переменной неправильные объекты. Зная, с каким объектом идет работа, компилятор может проверить код на предмет соответствия свойств и методов выбранному классу. Если используется динамическое связывание, компилятор не может выполнить подобную проверку. Если же программа попытается использовать несоответствующие свойства или методы, произойдет ошибка выполнения.
- **Б*о*льшая производительность.** Поскольку компилятор может заранее определить, имеет ли объект используемые в программе свойства и методы, программе не придется останавливаться для подобной проверки во время работы.
- **Более понятный код.** Посмотрев на объявление, можно сказать, какой класс объекта должна содержать переменная.

Несмотря на данные преимущества, в некоторых ситуациях лучше использовать динамическое связывание:

- может возникнуть необходимость в использовании одной и той же переменной для различных классов объектов. Это неплохой подход если различные классы имеют общие методы или свойства, которые нужно использовать в коде. Применяя такую переменную, не придется переписывать код, выполняющий одинаковые операции с разными объектами;
- некоторые объекты не могут определяться через раннее связывание, когда доступ к ним осуществляется через другое приложение. Использование объектов в других приложениях — новейшая технология VBA (см. главу 57).

Как известно, переменная типа `Variant` может содержать любой тип информации, включая ссылку на объект. Если переменная объявлена без четкого указания типа данных, VBA определяет ее как `Variant`. Программа также будет работать, но немного медленнее.

Присваивание переменной ссылки на объект

После определения объектной переменной, перед использованием, присвойте ей ссылку на определенный объект, применяя ключевое слово `Set`, как в данном примере:

```
Set objShapeObject = ThisDocument.Pages(1).Shapes(4)
```

Обратите внимание: синтаксис немного отличается от способа присвоения значений переменным других типов данных (см. главу 53). Как и с другими типами данных, между именем переменной и присваиваемым ей объектом помещается знак равенства. Разница заключаетсяв том, что в начале оператора используется ключевое слово `Set`.

Обнуление объектной переменной

Когда необходимость в доступе к объекту отпадает, рекомендуется уничтожить связи между объектом и переменной, с которой он ассоциирован. Таким образом, можно быть уверенным, что программа по ошибке не внесет изменения в объект. Техника достаточно проста. Используя ключевое слово `Set`, присвойте переменной значение `Nothing`, например:

```
Set objPriceIsNoObject = Nothing
```

Создание новых объектов

Если необходимого для работы объекта не существует, его следует создать. В простых программах VBA для этой цели используется метод `Add`. Метод `Add` применяют для создания встроенных объектов, доступных в VBA-приложении (с тем, с которым связан проект).

Использование метода Add

Фокус заключается в следующем: необходимо знать, с каким объектом можно использовать метод `Add`. Сначала следует найти контейнер, в котором содержится необходимый объект.

Предположим, нужно создать новый объект `Slide` в PowerPoint. Каждый объект `Slide` входит в объект `Slides`, библиотеку (из одного или нескольких) слайдов. Чтобы создать новый слайд, используйте метод `Slides.Add`. Конечно, так как при создании нового объекта `Slide` требуется определить объект `Slides`, в котором он будет создан, то полностью оператор будет выглядеть следующим образом:

```
ActivePresentation.Slides.Add 1, ppLayoutTextAndClipart
```

Даже если объект `Slide` (отдельный) имеет метод `Add` (на самом деле его нет), использовать его не удастся. Он будет создавать объект, дочерний `Slide`, а не другой объект `Slide`.

При создании объекта создавайте для него переменную

Когда возникают проблемы с вводом длинных ссылок на объекты (а они возникают при использовании метода `Add`), создайте переменную для нового объекта. Таким образом, можно использовать переменную вместо еще одной длинной ссылки на объект в любом месте кода. Два оператора, приведенных ниже, иллюстрируют этот подход:

```
Dim objMyBaby As Slide
objMyBaby = ActivePresentation.Slides.Add 1, _
    ppLayoutTextAndClipart
```

Создание объектов при помощи New и CreateObject

Иногда применяются другие способы создания новых объектов, например для:

- новых копий существующей формы в собственном проекте;
- объектов другого приложения или компонента ActiveX (COM);
- определенных объектов, которые основаны на классах, описанных пользователем в модулях класса.

В зависимости от ситуации, для создания объектов можно использовать ключевое слово `New` в определении переменной или операторах `Set` либо функцию `CreateObject`. Их действие становится ясным, если попробовать создать с помощью ключевого слова `New` или функции `CreateObject` обычные объекты.

В следующем примере демонстрируется использование оператора `Set` для создания нового объекта:

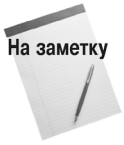
```
Set objCustomThermostat = New Thermostat
```

Создание эффективного кода работы с объектами при помощи оператора With

Не важно, как обращается пользователь к объекту; при помощи краткой, мнемонической переменной или длинного неразборчивого объектного выражения, набирая его с огромным трудом вновь и вновь. Можно этого не делать. Если программа использует один и тот же объект в двух или более последовательных операторах, оператор `With` позволит ввести имя объекта только однажды, что не только позволит избавиться от утомительного перепечатывания, но и сделать код быстрее и легче для понимания. Вот пример:

```
With objIHaveNoObjection
    .Name = "Последняя соломинка" ' Устанавливает свойство Name
    .DisplayName ' Вызывает метод DisplayName
    sngArea = .Area ' Получает значение свойства Area
    intStretchFactor = .Rotate (60) ' Вызывает метод Rotate,
    ' присваивая возвращаемое значение переменной
End With
```

Внутри структуры `With...End With` можно использовать операторы, которые проверяют и задают свойства, и операторы, вызывающие методы. Обратите внимание: структура `With` — это не цикл, содержащиеся в ней операторы выполняются только один раз.



Структуры `With` могут быть вложенными. Таким образом, можно выполнить несколько действий как над объектом, так и над другим содержащимся в нем объектом.

Сравнение объектных ссылок

Если использовать переменные для хранения объектных ссылок, возникает момент, когда может понадобиться проверка, не являются ли они ссылками на один и тот же объект. Для такой проверки используйте оператор `Is`. Значение выражения равно `True` если переменные указывают на один и тот же объект, и `False` если нет. Ниже приведен простой код, демонстрирующий использование `Is`:

```
Dim objObject1 As Object, objObject2 As Object
...
If objObject1 Is objObject2 Then
    MsgBox "Это один и тот же объект!"
Esle
    MsgBox "Это разные объекты!"
End If
```

Конечно, при помощи того же оператора `Is` можно также сравнивать объектные переменные с объектными выражениями:

```
If objObject3 Is ThisDocument.Pages(2).Shapes(3) Then
```

Обратите внимание: нельзя использовать `Is` (или любой другой оператор) для сравнения содержимого двух различных объектов.

Управление наборами данных при помощи библиотек объектов

Библиотека (Collection) — специальный тип объекта VBA. Как и предполагается из названия, роль библиотеки заключается в упрощении работы с набором данных как единым элементом. В обычной библиотеке все содержащиеся в ней объекты — одного типа. Библиотека `Shapes` в документе Office, например, содержит множество объектов `Shape`, а библиотека `Sections` в Word —множество объектов `Section`.

Некоторые библиотеки, однако, менее требовательны к содержащимся в них объектам. В VBA существует класс `Collection`. С его помощью можно создавать свои библиотеки объектов и использовать их для хранения данных и объектов любого типа в любом сочетании. Несмотря на определенные ограничения, у библиотек есть значительные преимущества перед массивами.

Библиотека может хранить простой список элементов как одномерный массив. Однако, принимая во внимание, что библиотеки могут считаться более общим типом массива, они могут помочь в выполнении сложных задач.

Работа с объектами Collection

Что следует знать про объекты `Collection`, так это способ доступа к отдельным элементам, содержащимся в них. Существуют такие возможности:

- сослаться на объект по его "месту" или *индексу* в библиотеке. В Word VBA, `Documents(2)` описывает второй объект `Document` в библиотеке `Documents`;
- сослаться на объект по имени. Многие объекты имеют имена, и имя нужного объекта можно использовать в объектном выражении. Например, панели управления в Office имеют свои имена, поэтому выражение типа `CommandBars("Debug")` определяет одну из панелей управления в библиотеке `CommandBars`.

С помощью этих подходов можно получить доступ к элементу, хранящемуся в библиотеке, но их нельзя использовать для присвоения нового значения существующему элементу библиотеки. Следующий оператор присваивания не работает:

```
colInventory(1465) = 119
```

Единственный способ изменить значение в созданной пользователем библиотеке VBA — удалить существующий элемент и добавить новый, содержащий нужное значение. Некоторые встроенные библиотеки VBA и приложений Office не имеют такого ограничения.

Компромиссы

При работе с объектами VBA можно прийти к выводу, что объекты `Collection` удобнее массивов в работе с наборами данных. Методы `Add` и `Remove` позволяют легко изменять размер библиотеки. Также вероятность возникновения ошибок при работе с ними гораздо меньше, чем при работе с оператором `ReDim`. (Эти преимущества особенно видны, когда нужно часто добавлять или удалять элементы из группы.)

Кроме того, можно давать элементам библиотеки имена и в дальнейшем получать доступ к ним по имени, что позволяет не только легко запоминать элементы, но и уменьшает время доступа к данным, особенно если библиотека содержит больше сотни элементов.

Создание объектов библиотек

Объект типа `Collection` создается в программе, как и любой другой объект, с использованием оператора `As` для определения типа объекта. Как и в случае с другими объектами, можно использовать два способа:

- объявить имя переменной объекта и затем использовать оператор `Set` для его создания. Заметим, что для создания новой библиотеки вместе с оператором `Set` нужно использовать ключевое слово `New`:

```
Dim colMixedBag As Collection
...
Set colMixedBAg = New Collection ' Создание библиотеки
colMixedBag.Add "Петя и Лена" ' Добавление элемента данных
```

- заставить VBA автоматически создать объект при первом использовании переменной в коде, для чего объявить переменную с ключевым словом `New`.

```
Dim colSetOfStuff As New Collection
...
' Следующий оператор создает библиотеку
' добавляя в нее целое число
colSetOfStuff.Add intStuffing
```

Добавление данных в библиотеку

После создания библиотеки для заполнения ее данными используется метод `Add`, так же как и во встроенных библиотеках VBA. Работу метода `Add` иллюстрировали предыдущие примеры. Полный синтаксис метода `Add` выглядит так:

```
Add (элемент[, ключ][, предыдущий индекс][, последующий
индекс])
```

Выражение для элемента обязательно. Оно может быть литералом, переменной, ссылкой на объект или сложным выражением, состоящим нескольких компонентов — в общем, всего, что возвращает значение, которое может понять VBA. Остальные параметры метода Add необязательны. Не будем рассматривать их все, однако термин "ключ" заслуживает более пристального внимания.

На элемент библиотеки можно сослаться по индексу, лучше дать ему значимое имя. Для этого укажите имя в виде строки при добавлении элемента в библиотеку:

```
colFinancials.Add 14323.44, "Продажи за февраль"
```

Предыдущий оператор добавляет значение 13323.44 в библиотеку colFinancials. В то же время он создает *ключ* или имя элемента.

В следующем примере встроенный объект Word ThisDocument (который ссылается на документ, содержащий текущий проект VBA) добавляется в библиотеку, и ему присваивается имя:

```
colGrabBag.Add ThisDocument, "DocKey"
```

Ключ не просто легче запомнить, чем числовой индекс, он является единственным надежным средством для быстрого доступа к отдельному элементу библиотеки. В связи со способом работы методов Add и Remove индексы отдельных элементов библиотеки могут изменяться. Но даже если 63-ий элемент станет 29-ым, к нему все равно можно будет обратиться по имени.

В библиотеках, основанных на родовом классе VBA Collection, единственным способом присвоения ключа является метод Add. Чтобы присвоить ключ элементу, у которого его нет, или чтобы изменить существующий ключ, добавьте элемент с нужным именем и удалите оригинал.

Удаление элементов

Метод Remove удаляет элементы из библиотеки. Определить элемент, который нужно удалить, можно по номеру или имени:

```
colMineral.Remove 2123
colMineral.Remove "Bauxite"
```

Помните, что при удалении элемента VBA заполняет оставшееся место, иначе говоря, индексы всех последующих элементов уменьшаются на единицу.

Подсчет элементов библиотеки

Очень легко потерять представление о размерах библиотеки, особенно если добавить или удалить несколько элементов. Родовой объект VBA Collection имеет только одно свойство, однако оно обладает исключительным значением — это свойство Count. Его можно присвоить переменной:

```
intCollectionSize = col20Questions.Count
```

или использовать в условном выражении:

```
If colPrices.Count > 1000
    MsgBox "У нас слишком много элементов!"
End If
```

Использование циклов For Each...Next для работы с библиотеками

Циклы `For Each...Next` применяются для работы со всеми элементами библиотеки поочередно. Вот их синтаксис:

```
For Each объектная_переменная In Collection
(операторы, которые должны выполняться при каждом проходе
цикла)
Next объектная_переменная
```

Главное отличие между структурой `For Each...Next` и стандартной `For...Next` заключается в том, что не нужно определять число прохождений цикла — VBA сделает это самостоятельно. Вместо переменной-счетчика, структура `For Each` требует переменной элемента, который отвечал бы типу объектов, хранящихся в библиотеке. Конечно, не забудьте также определить, с какой библиотекой нужно работать.

Следующий простой пример иллюстрирует работу `For Each...Next` в Word, PowerPoint или Excel, т.е. везде где есть библиотека `Shapes` объектов `Shape` (рисунков). Первая структура `For Each...Next` просто выводит в окне Immediate (Промежуточные значения) имя каждого элемента `Shape` из библиотеки `Shapes`. Второй цикл ищет объект с именем `WidgetA` и удаляет его:

```
' Вначале помещается ссылка на объект документа
' в переменную someDocument
Dim objS As Shape
For Each objS In someDocument.Shapes
    Debug.Print objS.Name
Next objS
For Each objS In someDocument.Shapes
    If objS.Name = "WidgetA" Then
       objS.Delete
       Exit For
    End If
Next objS
```

Оператор `Exit For` позволяет выйти из цикла сразу после нахождения нужного элемента в библиотеке. Если необходимо выполнить действия над каждым элементом коллекции, включать оператор `Exit For` не нужно.

Пользовательские объекты: модули классов

Когда встроенные объекты VBA достаточно изучены, можно приступать к созданию собственных. Хотя проявив чудеса ловкости при использовании процедур `Sub` и `Function`, вы можете добиться того, что обособление кода в объектах позволит получить серьезные преимущества.

- Хранение кода, работающего с наборами данных в одном объекте, позволяет уменьшить возможность появления ошибок при внесении изменений в программу.
- Программа становится более легкой для чтения и понимания.
- Можно создать сколько угодно независимых копий объекта, написав два оператора для каждой копии.
- Достаточно удобно использовать одинаковые свойства и методы с различными классами объектов (существует даже специальный технический термин — *полиморфизм*).

Как известно, объект состоит из определенных данных (свойств объекта) и кода, который изменяет эти данные (методы). Учитывая, что свойства — это переменные, а методы — те же процедуры, написание кода, который определяет объект, не будет сложным. Однако нужно следовать определенным правилам, чтобы VBA программа поняла, что именно пытается сделать пользователь.

Создание модулей классов

В VBA *класс* — это шаблон, на котором основан объект. Класс определяет, какие свойства, методы и события имеет объект и как себя ведет каждый из этих компонентов. Чтобы создать класс, вначале добавьте новый модуль класса в проект VBA (выбрав команду меню Insert⇨Class Module (Вставка⇨Модуль класса) или щелкнув правой кнопкой мыши в окне Project (Проект) и выбрав Insert (Вставить)). Окно модуля класса выглядит и действует так же, как и окно кода.

Компоненты определения класса

Обычный класс имеет три основных компоненты:

- закрытые объявления переменных, используемых объектом;
- открытые объявления процедур, которые позволяют процедурам из стандартных модулей возвращать или изменять значение свойства;
- открытые объявления процедур, которые определяют действия, выполняемые методами объекта.

Ниже приведено определение выдуманного класса `Thermostat`, в котором содержаться все три компоненты, перечисленные выше. Этот пример не совершает ничего полезного, но он работает — и это главное:

```
Private sngDegrees As Single '
' Код процедуры свойства Let Temperature:
Public Property Let Temperature(ByVal sngInput As _
    Single)
    sngDegrees = sngInput
End Property

' Код процедуры свойства Get Temperature:
Public Property Get Temperature() As Single
    Temperature = sngDegrees
End Property

' Код метода CalculateEnergyUse:
Public Sub CalculateEnergyUse()
    Const cstConversionFactor = 2.45
```

```
    Dim dblResult
    dblResult = sngDegrees * 365 * cstConversionFactor
    MsgBox "Годовой расход энергии этого термостата" & _
    " настройка рассчитана на " & dblResult & _
    " ватт"
End Sub
```

Объявление переменных класса

Для объявления переменных с целью доступа доступа к нескольким свойствам или методам используется раздел объявлений (Declarations) в верхней части модуля класса. Они всегда объявляются как `Private`. Задача объектов — не дать программе напрямую работать с данными. В этом разделе могут объявляться переменные, которые работают только с одним свойством или методом.

Как минимум нужно объявить по одной переменной для каждого из свойств объекта. Имя переменной не должно совпадать с именем свойства (далее показано, как определить имя свойства). Можно также объявить другие данные, которые будут использоваться внутри объекта и не будут доступны иным частям программы.

Написание процедур свойств

Чтобы добавить объекту свойство, нужно написать пару специальных процедур, — `Let` и `Get`. Обе процедуры в каждой паре должны иметь одинаковое имя.

Имя свойства — это имя, выбранное для процедур свойства `Let` и `Get`. Имя должно отражать содержание функции или процедуры.

И еще: при создании свойства, представляющего собой ссылку на объект, замените процедуру `Property Set` на процедуру `Property Let`, как половину пары процедур свойства. В противном случае, такие свойства будут действовать, как и свойства других типов данных.

Установка свойств объекта при помощи процедур Property Let

Процедура `Property Let` устанавливает значение свойства. В простейшей форме процедура `Property Let` принимает значение, передаваемое как параметр, и присваивает это значение переменной, которая представляет свойство:

```
Public Property Let Temperature (ByVal sngInput As _
    Single)
    sngDegrees = sngInput
End Property
```

Если в основной части программы введен оператор, устанавливающий свойство типа `Thermostat.Temperature = 75`, VBA вызовет процедуру `Let Temperature` с параметром `75`.

Конечно, процедуры свойств не ограничиваются одной строкой кода. Можно добавить операторы, которые будут проверять значение параметра, перед присвоением его свойству. В зависимости от параметра, можно предпринять еще какие-либо действия.

Возвращение свойств объекта при помощи процедур Property Get

Процедура `Property Get` работает так же, как и процедура `Function`; она возвращает значение — значение свойства. Как и в случае с процедурой `Function`, пользователь присваивает значение, которое должна возвращать функция, например:

```
Public Property Get Temperature() As Single
    Temperature = sngDegrees
End Property
```

Другие части программы могут вызывать процедуру `Get Temperature`, чтобы присвоить возвращаемое значение переменной или проверить возвращаемое значение при помощи условного выражения:

```
sngCurrentSetting = Thermostat.Temperature
```

или

```
If Thermostat.Temperature > 80
    MsgBox "Предлагается выключить нагреватель"
EndIf
```

Написание методов

Методы — просто обыкновенные процедуры `Sub` и `Function`, которые хранятся в модуле классов. В большинстве случаев, конечно, метод должен выполнять действия, связанные с объектом, изменяя при этом данные, хранимые объектом. VBA автоматически связывает методы с их классом, что дает возможность вызывать их из других частей программы так же, как и методы встроенных объектов.

Применение пользовательских объектов

Пользовательские объекты, основанные на пользовательских же классах, вызываются абсолютно так же, как встроенные объекты VBA и используемого приложения.

1. Объявите переменную для объекта, например:

```
objCustomThermostat = Thermostat
```

2. Используя оператор `Set`, создайте реальный объект, с которым можно работать:

```
Set objCustomThermostat = New Thermostat
```

3. Для доступа к свойствам или для выбора методов примените обычный "точечный" синтаксис VBA:

```
objCustomThermostat.Temperature = 65
objCustomThermostat.Calcu2lateEnergyCosts
```

4. Освободите память после завершения работы с объектом:

```
Set objCustomThermostat = Nothing
```