Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

О.Б. Калугина
М.В. Надеина
Г.И. Лукьянов

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Утверждено Редакционно-издательским советом университета
в качестве учебного пособия

Магнитогорск
2015

УДК 004.42(075)
ББК 32.973.26–018.7

Рецензенты:

Кандидат технических наук,
заместитель директора по развитию АСУ,
ЗАО «КонсОМ СКС»
Ю.И. Волщуков

Кандидат технических наук,
доцент кафедры вычислительной техники и программирования,
ФГБОУ ВПО «Магнитогорский государственный технический университет им. Г.И. Носова»
А.В. Леднов

Калугина О.Б., Надеина М.В., Лукьянов Г.И.
Объектно-ориентированное программирование [Электронный ресурс] : учебное пособие / Ольга Борисовна Калугина, Маргарита Владимировна Надеина, Георгий Игоревич Лукьянов ; ФГБОУ ВПО «Магнитогорский государственный технический университет им. Г.И. Носова». – Электрон. текстовые дан. (1,19 Мб). – Магнитогорск : ФГБОУ ВПО «МГТУ», 2015. – 1 электрон. опт. диск (CD-R). – Систем. требования : IBM PC, любой, более 1 GHz ; 512 Мб RAM ; 10 Мб HDD ; MS Windows XP и выше ; Adobe Reader 8.0 и выше ; CD/DVD-ROM дисковод ; мышь. – Загл. с титул. экрана.

Учебное пособие позволяет освоить теоретические и практические аспекты объектно-ориентированного программирования на примере двух объектно-ориентированных языков — VBA и Delphi. Особое внимание уделено основным принципам объектно-ориентированного подхода — инкапсуляции, наследованию и полиморфизму, а также особенностям иерархии объектной модели, в зависимости от того, в какое приложение встроен язык VBA. Пособие содержит основные теоретические и практические сведения по технологии объектно-ориентированного и визуального программирования и предназначено для эффективного и быстрого освоения основных методов создания пользовательских приложений.

Пособие предназначено для бакалавров и магистров очной и заочной форм обучения по направлению 080200.62 Менеджмент.

УДК 004.42(075)
ББК 32.973.26–018.7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Концепция компьютерного программирования, известная как ООР (object-oriented programming) была разработана в середине восьмидесятых годов. В переводе с английского object-oriented programming означает объектно-ориентированное программирование (сокращенно ООП). Основная идея объектно-ориентированного программирования заключается в объектах, как составных элементах программных приложений.

Создавать объектно-ориентированный код сложно, а создавать код, используемай многократно еще сложнее. Необходимо выбрать объекты на роли классов, установить взаимоотношения между ними, определить структуру наследования, выделить интерфейсы и т.п. Дизайн программы должен отражать предметную область проблемы с одной стороны, но быть обобщенным с другой стороны, чтобы оставить возможности для удовлетворения требований, которые могут возникнуть в будущем. Так же важной задачей является отсутствие необходимости или, по крайней мере, минимизация необходимости изменения дизайна программы. Опытные программисты подтвердят вам, что идеальный дизайн практически недостижим при первом проходе. Необходимо вникнуть в предметную область, выявить подводные камни и т.п. Нередко, первый проход при построении программы выделяют в отдельную задачу прототипирования. Перед тем как перейти непосредственно к рассмотрению объектно-ориентированного программирования, рассмотрим историю развития парадигм программирования в общем.

Парадигмы программирования

Парадигма программирования — это система идей и понятий, определяющих стиль написания компьютерных программ, а также образ мышления программиста.

Т.е. парадигма программирования определяет то, какими понятиями вы оперируете и каких правил придерживаетесь при написании кода.

Выделяют три основные парадигмы программирования:

- процедурное программирование
- объектно ориентированное программирование
- обобщенное программирование

В приложении 1 приведен неполный список объектно-ориентированных языков программирования. Важно не забывать, что парадигма программирования не определяется используемым языком. Язык и его синтаксические возможности могут лучше поддерживать одну парадигму и хуже – другую, однако, с большой вероятностью мы сможем симулировать использование парадигмы программирования на языке, для этого не предназначенном. В качестве примера можно рассмотреть использование C++, как языка процедурного программирования и наоборот – C – как языка объектно-ориентированного программирования. Скорее всего язык поддержит парадигму не полностью, но все же частично поддержит. Рассмотрим основные парадигмы программирования.

Процедурное программирование

Процедурное программирование является отражением архитектуры традиционных ЭВМ, предложенной ***Фон Нейманом*** в сороковых годах. Теоретической основой этой парадигмы является конечный автомат Тьюринга. Машина Тьюринга – это аппарат, способный за конечное время выполнять алгоритмы пошагово с помощью элементарных действий. Суть процедурного программирования сводится к последовательному выполнению операторов с целью преобразования состояния оперативной памяти. Т.е. программа в рамках своего жизненного цикла последовательно обновляет память результатами своей работы. Языки, поддерживающие процедурную парадигму программирования предоставляют

возможность программисту определять каждый шаг выполнения программы. Программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.

ОСНОВНЫЕ ПОНЯТИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

В основе концепции объектно-ориентированного программирования лежит понятие объекта — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

Например, объект Студент может иметь следующие

данные (свойства, поля):

- Фамилия,
- Имя,
- Отчество,
- Год рождения,
- Пол,
- Дата поступления,
- код студенческой группы и др.

Методы:

- Читать научную литературу,
- Выполнять письменные задания,
- Посетить занятие,
- Сдать лабораторную работу,
- Сдать курсовую работу,
- Сдать зачет,
- Сдать экзамен

И может обрабатывать с помощью доступных ему методов наступление следующих событий:

- Наступило время аудиторного занятия
- Получено домашнее задание
- Получено задание для курсовой работы
- Наступило время экзамена
- Осталось 3 дня до экзамена
- Наступило время зачета
- Получена отличная оценка

и другие.

Объект

Объектом называется любая именованная сущность, имеющая:

- свойства, то есть установки, которые можно проверить и изменить;
- методы, то есть действия, которые может выполнить объект, когда программа попросит об этом;
- события, то есть ситуации, в которых объект оказывается и на которые может ответить заранее определенными для таких ситуаций действиями.

Свойства

Свойства - это характеристики объекта. Каждое свойство хранит информацию о некотором аспекте внешнего вида, поведения, содержимого объекта. Главной задачей свойства является описание некоторой характеристики объекта.

Методы

Методы - это именованные действия, которые объект может выполнить по команде. Ввиду того, что любой метод является неотъемлемой частью объекта, объект сам знает, что ему делать, когда вызывается метод. Таким образом, методы

- не что иное, как процедуры, привязанные к конкретному объекту. Чтобы вызвать метод, необходимо напечатать имя объекта, точку, а затем имя метода.

События

Событие представляет собой нечто, случающееся с объектом, и то, на что объект может ответить заранее предусмотренным действием. К событиям можно отнести следующее:

физические действия пользователя программы, например щелчок кнопкой мыши, перемещение курсора и так далее;ситуации, в которые попадает объект в ходе выполнения программы.

Появление ОО концепции

Первым языком программирования, в котором были предложены принципы объектной ориентированности, был язык Симула. В момент его появления в 1967 году в нём были предложены революционные идеи: объекты, классы, виртуальные методы и др., однако это всё не было воспринято современниками как нечто грандиозное. Тем не менее, большинство концепций были развиты Аланом Кэем и Дэном Ингаллсом в языке Smalltalk. Именно он стал первым широко распространённым объектно-ориентированным языком программирования.

В настоящее время количество прикладных языков программирования (приложение 1), реализующих объектно-ориентированную парадигму, является наибольшим по отношению к другим парадигмам. В области системного программирования до сих пор применяется парадигма процедурного программирования, и общепринятым языком программирования является Си. При взаимодействии системного и прикладного уровней операционных систем заметное влияние стали оказывать языки объектно-ориентированного программирования.

Особенности объектно-ориентированной парадигмы

В современных ОО языках используются механизмы:

- **Наследование.** Мощь ООП проявляется в механизме наследования: классы могут наследовать поля и методы других классов и добавлять к ним свои. Это позволяет ускорить процесс разработки за счет использования уже проверенного, готового кода. Некоторые ОО языки позволяют выполнять множественное наследование, то есть объединять в одном классе возможности нескольких других классов.
- **Инкапсуляция.** Если язык поддерживает инкапсуляцию, то это означает, что он предоставляет какой-либо механизм объединения данных и кода обработки этих данных (функций, методов и т.п.) в объект. Но далеко не все поля и методы должны быть видимы снаружи. Многие из них используются только внутри объекта и потому их лучше скрыть. Объект скрывает приватные (**private**) поля и методы от внешнего доступа.

  Сокрытие деталей реализации, которое позволяет вносить изменения в части программы безболезненно для других её частей, что существенно упрощает сопровождение и модификацию ПО.
- **Полиморфизм.** При полиморфизме некоторые части (методы) родительского класса заменяются новыми, реализующими специфические для данного потомка действия. Таким образом, интерфейс классов остаётся прежним, а реализация методов с одинаковым названием и набором параметров различается. В ООП обычно применяется полиморфизм подтипов (называемый при этом просто «полиморфизмом»), нередко в форме позднего связывания.

ОБЪЕКТЫ В VBA. ТИПЫ И КЛАССЫ

Класс является обобщением понятия типа данных и задает свойства и поведение объектов класса - экземпляров класса. Каждый объект принадлежит некоторому классу. Отношение между объектом и его классом такое же, как между переменной и ее типом. Класс - это объединение данных и обрабатывающих их процедур и функций. Данные называются также переменными класса, а процедуры и функции - методами класса. Переменные определяют свойства объекта, а совокупность их значений - состояние объекта. Наряду со свойствами и методами с классом связывается еще одно понятие - события. Каждый класс имеет определенный набор событий, которые могут возникать при работе с объектами класса, - чаще всего при определенных действиях пользователя, иногда, как результат действия системы. При возникновении события, связанного с тем или иным объектом, система посылает сообщение объекту, которое может быть обработано методом - обработчиком события, специально созданным при конструировании объекта. События обеспечивают большую гибкость при работе с объектами. Методы класса выполняются одинаково для всех объектов класса, а на события каждый объект реагирует индивидуально, поскольку имеет собственный обработчик события.

Программирование в классах

На объектах построена операционная система Windows, - окно являются ее основным объектом. Сам Office полностью построен на классах и работе с объектами этих классов, - здесь все, начиная от приложения и кончая отдельным символом, рассматривается как объект некоторого класса. Профессиональный прикладной программист, работающий в некоторой проблемной области и решающий разнообразные задачи из этой области, как правило, начинает с создания классов, описывающих специфику данной проблемной области. Затем уже решение тех или иных специальных задач он описывает в терминах работы с объектами данной проблемной области.

VBA позволяет программисту создавать собственные классы. Синтаксически класс представляет отдельный модуль специального вида - модуль класса. Мы постараемся сейчас детально и на примерах разобраться во всех особенностях этой важной конструкции.

Пример создания класса

Операции, которые можно определить для этого типа:

`InitPerson` - инициализация полей записи;

`PrintPerson` - печать полей записи;

`CopyPerson(Source As Person)` - копирование источника (записи `Source`) - аналог оператора присвоения;

`WhoIs` - более специфическая операция, с определенной достоверностью определяющая пол, анализируя имя и фамилию.

Всякий раз при определении пользовательского типа, так или иначе, но следует определить операции над данными этого типа. Естественно собрать определение типа и операции над ним в одном месте. Такое объединение представляет уже почти полное, с точностью до событий определение типа. Для типов, определенных подобным образом, введен новый термин - **класс** .

Синтаксически классы в VBA оформляются в виде **модуля класса** . Поэтому начинать создание класса в Редакторе Visual Basic нужно с выбора в меню Insert пункта Class Module. Этот модуль имеет такую же структуру, как и стандартный модуль, о котором мы подробно уже рассказали. Модуль состоит из двух разделов - объявлений и методов. В первом из них естественным образом описываются свойства класса, а во втором - его методы. И здесь действуют спецификаторы области действия `Public` и

Private.Public - свойства и Public -методы составляют интерфейс класса. Только к этим свойствам и методам можно обращаться при работе с объектами класса, объявленными в других модулях, где класс является видимым. Часть из этих операций уже упомянута, другие ясны из контекста. Приведем с самого начала полное описание класса, а потом уже, когда полная картина будет ясна, перейдем к деталям. Создав модуль класса Личность, мы поместили туда следующий текст:

```
Option  Explicit

'Класс  Личность
'Свойства  класса:  имя,  отчество,  фамилию,  дату  рождения
'закроем  от  прямого  доступа,
'получить  и  изменить  их  можно  только  через  методы  класса
Private  Имя  As  String
Private  Отчество  As  String
Private  Фамилия  As  String
Private  ДатаРождения  As  Date

Public  Sub  InitPerson(ByVal  FN  As  String,  ByVal  LN  As
String,    _
                        ByVal  DoB  As  Date)
        'Инициализация  личности
        Имя  =  FN
        Фамилия  =  LN
        ДатаРождения  =  DoB
End  Sub

Public  Sub  PrintPerson()
        'Печать  в  отладочном  окне  Immediate
        Dim  S  As  String
        If  WhoIs  Then  S  =  "родилась"  Else  S  =  "родился"
        Debug.Print  Имя,  Отчество,  Фамилия,  S,  ДатаРождения
End  Sub

Public  Sub  CopyPerson(You  As  Личность)
        Имя  =  You.ВашеИмя
        Фамилия  =  You.ВашаФамилия
        ДатаРождения  =  You.ВашаДатаРождения
End  Sub

Public  Function  WhoIs()  As  Boolean
        'Пытается  определить  пол  личности,  анализируя  имя  и
фамилию
        'Возвращает  True,  если  думает,  что  имеет  дело  с
женщиной.
```

```
        Dim F1 As Boolean, F2 As Boolean
        F1 = ПоследняяБуква(Имя) = "А" Or
ПоследняяБуква(Имя) = "Я"
        F2 = ПоследняяБуква(Фамилия) = "А" Or
ПоследняяБуква(Фамилия) = "Я"
        If F1 And F2 Then
                'можно полагать, что наша Личность -
женщина
                WhoIs = True
        ElseIf Not F1 And Not F2 Then
                WhoIs = False
        Else 'Есть сомнения
                If Отчество = "" Then
                        Отчество = InputBox(Имя & " " &
Фамилия _
                        & "! " & "Назовите отчество,
пожалуйста.")
                End If
                WhoIs = ПоследняяБуква(Отчество) = "А"
        End If
End Function

Public Sub SayWhoIs()
'       Вывод сообщения о поле и возрасте личности
        If WhoIs Then
                MsgBox ("Думаю," & Имя & _
                                ", Вы из прекрасной
половины человечества!")
        ElseIf Year(ДатаРождения) > 1967 Then
                MsgBox ("Думаю, " & Имя & ", Вы -
молодой человек!")
        Else
                MsgBox ("Думаю, " & Фамилия & ", -
мужчина!")
        End If
End Sub

Private Function ПоследняяБуква(ByVal W As String) As
String
        'Внутренняя функция: возвращает в верхнем регистре
                        'последнюю букву слова W
        ПоследняяБуква = UCase(Right(W, 1))
End Function

Public Property Get ВашеИмя() As String
```

```
        ВашеИмя = Имя
End Property

Public Property Let ВашеИмя(ByVal vNewValue As String)
        Имя = vNewValue
End Property
Public Property Get ВашеОтчество() As String
        ВашеОтчество = Отчество
End Property

Public Property Let ВашеОтчество(ByVal vNewValue As String)
      Отчество = vNewValue
End Property

Public Property Get ВашаФамилия() As String
        ВашаФамилия = Фамилия
End Property

Public Property Let ВашаФамилия(ByVal NewValue As String)
        Фамилия = NewValue
End Property

Public Property Get ВашаДатаРождения() As Date
        ВашаДатаРождения = ДатаРождения
End Property

Public Property Let ВашаДатаРождения(ByVal NewValue As
Date)
        ДатаРождения = NewValue
End Property

Private Sub Class_Initialize()
        Имя = "Адам"
        Фамилия = "Человек"
        ДатаРождения = #1/1/100#
End Sub
```

С точностью до имен все свойства класса `Личность` совпадают с полями типа `Person`. Кстати, мы сделали все свойства закрытыми (`Private`), и теперь вне класса нет прямого доступа к его полям (свойствам).

В класс `Личность` добавлено 5 общих методов: `InitPerson`, `PrintPerson`, `CopyPerson`, ***`WhoIs`***, `SayWhoIs` и по паре методов `Get` и `Let` на каждое закрытое свойство. В классе есть и закрытая для внешнего использования функция "`ПоследняяБуква` ", и обработчик события `Initialize`.

Метод Init в том или ином виде должен быть определен в каждом классе. Это первый вызываемый метод объекта. Прежде чем начать работу с объектом, его нужно инициализировать. В нашем классе эту работу и делает метод InitPerson.

Метод Print также присутствует почти в каждом классе - нужно же распечатать информацию об объекте! - в PrintPerson вызывается метод *WhoIs*.

CopyPerson - еще один общий, часто необходимый метод, позволяющий реализовать настоящее присвоение, когда копируется не ссылка, а значения полей класса, что позволяет иметь не две ссылки на один объект, а два идентичных объекта.

Булева функция *WhoIs* - метод, специфический для нашей задачи. Это попытка определить пол по имени и фамилии. Применяется примитивный, но обеспечивающий высокую достоверность для русских имен и фамилий алгоритм. По ходу дела потребовалось ввести вспомогательную функцию " ПоследняяБуква".

Метод SayWhoIs вызывает *WhoIs*, дополнительно определяет возраст (только для мужчин) и выводит соответствующее сообщение в окно Message.

Закрытая для внешнего использования функция ПоследняяБуква возвращает в верхнем регистре последнюю букву слова. Эта задача решается двумя вызовами функций работы со строками Right и UCase. Первая возвращает "хвост" слова, вторая - преобразует результат в верхний регистр.

Подробнее скажем о закрытых свойствах и специальных методах Get и Let. Вообще говоря, свойства можно не закрывать, а специальные методы не вводить. Но VBA позволяет следовать традициям объектно-ориентированного программирования, согласно которым считается правильным не давать возможности непосредственного изменения свойств, поскольку иногда это может привести к некорректному состоянию объекта. Поэтому доступ к свойствам закрывается, для чего достаточно объявить их с атрибутом Private, но зато вводятся специальные методы Get (для получения значения свойства) и Let (для изменения значения на новое). Заготовки для этой пары свойств строятся автоматически, если Вы при вставке метода указали (пометив флажок Property), что он должен быть свойством.

С каждым из объектов созданного класса связываются два события: Initialize и Terminate. Первое возникает при первоначальном обращении к объекту, второе - по окончании работы с ними. В приведенном примере в методе Initialize объект получает полную инициализацию, восходящую к «объекту-родителю».

Завершим изложение примером работы с объектами класса, его методами и свойствами:

```
Public  Sub  Знакомство()
        Dim  UserOne  As  New  Личность
        Dim  UserTwo  As  New  Личность
        Dim  UserThree  As  New  Личность
        Debug.Print  UserOne.ВашеИмя
        UserOne.InitPerson        FN:="Петр",        LN:="Петров",
DoB:=#1/23/1968#
        UserTwo.InitPerson        FN:="Анна",        LN:="Козлова",
DoB:=#7/21/1968#
        UserOne.PrintPerson
        UserTwo.PrintPerson
        UserOne.SayWhoIs
```

```
        UserTwo.SayWhoIs
        UserTwo.ВашаФамилия = UserOne.ВашаФамилия & "а"
        Debug.Print UserOne.ВашаФамилия
        Debug.Print UserTwo.ВашаФамилия
        UserThree.InitPerson FN:="Анна", LN:="Агеева",
DoB:=#5/17/1803#
        UserThree.PrintPerson
        UserThree.SayWhoIs
End Sub
```

Вот какие результаты отладочной печати будут выданы в окно Immediate (отладки):

```
Адам
Петр        Петров      родился          23.01.68
Анна        Козлова     родилась         21.07.68
Петров
Петрова
Анна      Агеева          родилась          17.05.1803
```

Объекты и переменные

Обобщим сведения об объектах и переменных Если есть описания переменных:

```
Dim X As T, Y As T1
```

то без контекста понять нельзя, что есть `X` и `Y` - "обычные" переменные или объекты. Например, если `T` - тип, заданный пользователем, а `T1` - определенный им *класс*, то `X` - это *переменная*, а `Y` - *объект*. С нашей точки зрения, *класс* и тип - понятия, если не эквивалентные, то близкие по смыслу. Также близки понятия объекта и переменной. **Класс - это специальная форма определения типа**. Если есть тип **`T`** и *класс* `T1`, то можно объявить *произвольное* число экземпляров типа `T` и класса`T1`. Экземпляры типа `T` называются переменными, класса **`T1`** - объектами. *Класс* задает свойства, методы и события своих объектов. Тип всегда задает свойства, неявно - методы, но никогда - события.

В объявлении: **`Dim Y As T1`**, где **`T1`** - *класс*, *объект* `Y` мы часто называем переменной, говоря, что она имеет тип `Object`. Переменные типа `Object` рассматриваются как ссылки, задающие *адрес* памяти, где хранится *объект*. У них фиксированная *длина* 4 байта. При объявлении такой переменной *память* для самого объекта может и не отводиться в отличие от обычных переменных, поэтому не определено и *значение* ссылки. Задать ссылку - связать переменную с объектом - можно двумя путями:

1. создать новый объект, выделив ему память;
2. сослаться на уже существующий объект.

Есть два способа связывания: **раннее** и **позднее**. При позднем связывании *переменная* объявляется так:

```
Dim V As Object
```

Это объявление говорит о том, что *переменная* является объектом (ссылкой), но ничего о классе этого объекта - он может быть произвольным, и выяснится это только динамически при выполнении программы, когда *переменная* `V` будет связываться с только что созданным или существующим объектом того или иного класса. Поэтому такое *связывание* называется поздним, или динамическим. Естественно, что необъявленные переменные или те, которым тип при объявлении не задан (значит, у них по умолчанию тип `Variant`), допускают только позднее *связывание*. При раннем связывании в момент объявления указывается *класс объекта*, например:

```
Dim Петров As Личность, Козлов As Личность
```

Это позволяет еще на этапе трансляции проверять, допустимы ли те или иные *операции* над объектами `Козлов` и `Петров`. За исключением весьма сомнительного преимущества позднего связывания - возможностью связаться с объектом любого типа, ранее *связывание* предпочтительнее во всех отношениях. Для программиста особенно важно, что при раннем связывании, когда задан *класс объекта*, в процессе работы с этим объектом идет подсказка о его свойствах и методах. Подсказка облегчает жизнь программиста, и лишаться ее не стоит. Поэтому наш совет - выбирайте раннее *связывание*.

Поясним, как создаются новые объекты, и как происходит *связывание* с объектами, уже существующими. Для удобства разделим все объекты на три группы:

1. объекты, чей класс определен пользователем в одном из модулей класса; например, объекты класса `Личность`, созданного в этой лекции;
2. объекты родового приложения (Excel, Word, PowerPoint), которому принадлежит проект и которые доступны по умолчанию;
3. `ActiveX` и `Com AddIns` -объекты, в частности, объекты других приложений при их подключении к исходному приложению. Например, в Word можно подключить объекты Excel, а документы Word включить в рабочие листы Excel.

Объекты, класс которых определен пользователем.

Понятно, что при работе с такими объектами хотя бы часть из них необходимо первоначально создать, и делается это при их объявлении:

```
{Dim | Private | Public | Static }<имя переменной> As New <имя типа>
```

Спецификатор `New` указывает, что в момент объявления нужно создать новый объект, то есть выделить ему память. В этот же момент ссылка на объект получает конкретное значение. Выделение памяти еще не означает инициализации значений свойств объекта. Инициализацию можно задать при определении события `Initialize` или в специально построенном методе `Init`, который следует запускать в начале работы с объектом. Вот пример создания объекта с последующей его инициализацией:

```
Public Sub MyCreateObject()
    'Нельзя создавать объекты собственного класса, используя CreateObject
    Dim Nemo As Личность
    'Set Nemo = CreateObject("Личность")
```

```
    'Можно их создавать, используя New
    Set Nemo = New Личность
    Nemo.InitPerson FN:="Prince", LN:="Dakar", DoB:=#1/23/1838#
    Nemo.PrintPerson
End Sub
```

Только метод `New` позволяет создать новый объект для классов, определенных программистом. Для этой цели нельзя, например, использовать метод `CreateObject`. Но, заметьте, спецификатор `New` не является обязательным, его можно опускать при объявлении таких переменных класса, которые в дальнейшем получат значение, благодаря ссылке на другие, ранее созданные объекты. Задать ссылку на существующий объект нельзя обычным оператором присвоения - для этого в VBA есть специальный оператор `Set`:

```
Set  <имя переменной - объекта> = {<объектное выражение>| Nothing
}
```

`Set` - специально выделенный, частный случай оператора присвоения, в его левой части стоит имя переменной, определенной как объект при объявлении, а в правой - некоторое выражение, значение которого - ссылка на объект. В итоге переменная левой части получает значение выражения правой части. Вот пример некоторого объявления объекта `Q`. Следующие строки мы добавили в конец предыдущей процедуры:

```
Dim Q As Личность
Set Q = Nemo
Q.PrintPerson
```

В результате будет отпечатан следующий текст:

```
Prince Dakar родился 23.01.1838
```

Объекты "родного" приложения

Каждый VBA-проект погружен в "родное" приложение Office, объекты которого доступны в проекте. Для создания таких объектов не используется ни спецификатор `New`, ни метод `CreateObject`. Новые объекты, если и создаются, то специальными методами своего класса. Например, метод `Add` позволяет добавлять элементы в коллекции. При работе с объектами этой группы объявляются переменные соответствующего класса, затем они связываются с уже существующими объектами. Связывание выполняет оператор `Set`. Все примеры из первой лекции книги посвящены работе с объектами этой группы. Напомним тот, где участвует один из наиболее употребительных объектов - `Range`:

```
Public Sub FirstComment()
Dim myRange As Range
'Добавление комментария
With ActiveDocument
    Set myRange =.Paragraphs(1).Range
   .Comments.Add myRange, "Эта лекция рассказывает о классах
объектов"
    'Передвигается объект Range
    myRange.Move Unit:=wdParagraph, Count:=2
```

```
End With

End Sub
```

Здесь вначале объявляется объект myRange класса Range, что обеспечивает раннее связывание. Объектное выражение в правой части оператора Set возвращает ссылку на существующий объект (первый параграф активного документа), и эта ссылка становится значением myRange. А затем с этим объектом можно работать, используя свойства и методы соответствующего класса, в данном случае - Range. Его можно использовать, например, в методе Add при работе с коллекцией комментариев или вызвать метод Move для изменения области, отведенной объекту.

ActiveX-объекты

Технология ActiveX обеспечивает взаимодействие приложений - Автоматизацию (Automation), при которой одно приложение управляет работой другого. На этой технологии построено взаимодействие приложений Office.

Чтобы начать работу с ActiveX -объектом, нужно объявить соответствующую переменную, создать сам объект и связать переменную с объектом. Один из способов выполнения этой работы - использовать спецификатор New в операторе объявления переменных, который в этой ситуации имеет вид:

```
Dim  <имя переменной>  As New  <имя приложения.имя класса>
```

По сравнению с обычной конструкцией оператора объявления здесь более сложно задается тип (класс) создаваемого объекта. Первая часть - имя_приложения - указывает приложение, создающее объект, и в то же время это имя библиотеки типов данного приложения - TypeLib. Приложение является сервером, который вызывается для обеспечения работы с объектом. Вторая часть -имя_класса - задает тот конкретный класс, определенный в сервере, экземпляр ActiveX -объекта которого будет создан. Библиотека типов подключаемого приложения должна быть видимой, для чего надо включить ссылку на нее в меню References. В следующем примере Excel подключается в приложении Word:

```
Public Sub WorkWithActiveX()
    'Создание объекта Excel.Application при работе с документом
Word
    Dim xlApp As Excel.Application
    'Set xlApp = CreateObject("Excel.Application.9")
    Set xlApp = New Excel.Application
    xlApp.Application.Visible = True
    'Добавить новую книгу
    xlApp.Workbooks.Add
    'Теперь можно работать и с ячейками данной книги
    xlApp.Workbooks(1).Activate
    xlApp.Range("A1:A2") = 2
    xlApp.Range("B1") = "=A1+A2"
    xlApp.Range("B2") = "=A1*A2"
    'Закрываем открытую книгу без сохранения изменений
```

```
    xlApp.Workbooks(1).Close SaveChanges:=False
    'Закрываем приложение
    xlApp.Quit
End Sub
```

Создание Excel приложения в данном случае возможно двумя способами - применением конструкции `New` или функции `CreateObject`. Однако, использование `New` для создания `ActiveX` -объектов не всегда возможно. Не все приложения допускают такой способ подключения, и не все объекты доступны, - например, недоступны внутренние `ActiveX` -объекты, которые совместно могут храниться в одном и том же файле. Более универсальный способ создания `ActiveX` -объектов - вызов специальных функций `CreateObject` и `GetObject`. Как правило, вызов этих функций помещается в правую часть оператора `Set`. В результате его выполнения объектное выражение возвращает ссылку на созданный объект, которая и становится значением переменной.

Синтаксис функции `CreateObject`:

```
CreateObject(<класс>)
```

Параметр `класс` - строка, задающая класс объекта, у которой две обязательные части: "имя_приложения.имя_класса"; ее вид совпадает с видом задания класса `ActiveX` -объектов в операторе объявления.

Функция `GetObject` имеет дополнительный параметр:

```
GetObject(<путь>, <класс>)
```

Параметр `путь` задает полный путь к файлу, содержащему `ActiveX` -объект. Этот объект и активизируется. Второй параметр при этом необязателен. Но если файл содержит группу `ActiveX` -объектов разных классов, задать его необходимо. Можно опустить и первый параметр, если `ActiveX` -объект этого класса уже создан, и тогда возвращается ссылка на ***активный объект*** заданного класса. Вот пример, в котором читается книга Excel, используя метод `GetObject`. Обратите внимание, книга создается без предварительного создания объекта `Application`:

```
Public Sub WorkWithGetObject()
    Const MY_PATH As String = "e:\O2000\CD2000\Ch4\"
    Dim myBook As Excel.Workbook
    Set myBook = GetObject(MY_PATH & "BookOne.xls")
    With myBook
       .Application.Visible = True
        'Чтобы увидеть книгу, включите ее в окне UnHide из меню
Window
       .Charts("Динамика Продаж").Activate
        'Посмотрев на диаграмму, переключимся на рабочий лист
        MsgBox ("Вы можете перейти в Excel")
       .Worksheets(1).Activate
       .Worksheets(1).Range("A40") = 777
       .Save
       .Application.Quit
    End With
End Sub
```

Модуль класса

Построим еще один простой класс Rational, определяющий рациональные числа. Начнем построение класса, как обычно, с комментария, содержательно описывающего назначение класса, его свойства и поведение. Вот комментарий, к нашему классу:

```
' Класс Rational
'Определяет новый тип данных - рациональные числа и основные
'операции над ними - сложение, вычитание, умножение и деление.
'Рациональное число задается парой целых чисел (m,n) и
изображается,
'обычно, в виде дроби m/n. Число m называется числителем, а
'n - знаменателем.  Для каждого рационального числа существует
'множество его представлений, например, - 1/2, 2/4, 3/6, 4/8, …
'задают одно и тоже рациональное число. Среди всех представлений
'можно выделить то, в котором числитель и знаменатель
несократимы.
'Именно такие представители будут храниться в нашем классе.
'Операции над рациональными числами определяются естественным
'образом. Лучшим их описанием будут соответствующие им методы.
```

После комментария следует описание переменных, задающих свойства класса. Это могут быть терминальные свойства, заданные обычными переменными VBA, как, например, Имя, Фамилия в классе " Личность ", так и свойства - участники. Напомним, что свойства - участники это объекты других классов. Без них не обойтись, если мы строим семейство классов. Помните, что в Office нет наследования классов в классическом понимании, - его заменяет встраивание. Так что при построении семейства классов, приходится иметь дело с "толстыми" объектами, свойства которых являются объектами, имеющими свойства, являющиеся объектами и так далее. В разбираемом нами случае речь не идет о семействе классов, поэтому класс Rational имеет только терминальные свойства. Пояснять, что это за свойства нет необходимости. Вот их определение:

```
'Свойства класса Rational
Private m As Integer     'числитель
Private n As Integer     'знаменатель
```

Сокрытие свойств

В приведенном примере свойства объявлены с описателем Private, а не Public . Таким же образом описаны свойства и в классе " Личность ". Это общепринятая практика объектно-ориентированного программирования. Если бы свойства имели атрибут Public, - были бы открытыми, тогда при работе с объектом, можно было бы иметь к ним прямой доступ, как при чтении, так и записи. Но такая свобода, как правило, недопустима. Вот возможные стратегии при работе со свойствами:

- Чтение, запись (Read - Write).
- Чтение, запись при первом обращении (Read, Write-once).
- Только чтение (Read-only).
- Только запись (Write-only).
- Ни чтение, ни запись (Not Read - Not Write), - свойство закрыто.

Открытость свойств не позволяет реализовать только первую из стратегий. В приведенном примере с классом Rational открытость свойств недопустима, поскольку

числитель и знаменатель преобразуются скрытым от внешнего пользователя способом, позволяющим всегда задавать единственного представителя рационального числа.

Замечание: При задании свойств делайте их закрытыми.

Для доступа к закрытым свойствам и реализации перечисленных выше стратегий работы с ними предложены специальные методы - свойства `Property Let`, `Set` и `Get`, примеры их мы видели в классе " `Личность` ", а подробный разговор еще предстоит.

Конструкторы и деструкторы. Стандартные события

Следующим шагом в создании класса является разработка его конструкторов и деструктора. Напомним, что новый объект - экземпляр класса создается конструкцией `New`, например:

```
Dim MyRationalNumber As New Rational
```

В этот момент:

- создается *типизированный указатель* `MyRationalNumber`,
- создается новый объект - экземпляр класса `Rational`,
- при создании объекта вызывается конструктор по умолчанию, инициализирующий этот объект, определяющий значения его свойств,
- указатель связывается с объектом.

В языках объектного программирования, как, например, в языке C++ конструкторов может быть несколько, среди которых выделяется конструктор без параметров, являющийся конструктором по умолчанию. Остальные конструкторы имеют параметры, позволяющие задать свойства объекта в момент инициализации. Важно понимать роль конструкторов особенно в ситуации, когда создается "толстый" объект. В момент создания такого объекта будет вызвана цепочка конструкторов, начиная с самого внутреннего, создающего самый внутренний объект, затем будет создан охватывающий объект, пока не будет вызван внешний конструктор, создающий объект - оболочку. В таком конструкторе приходится задавать параметры для инициализации всех внутренних объектов.

В VBA все проще. Во многом это объясняется тем, что здесь, в отличие от многих других языков программирования, есть разумная стратегия начальной инициализации переменных, - мы о ней говорили ранее. Поэтому здесь есть только конструктор по умолчанию - конструктор без параметров, да и тот часто не определяется, полагаясь на стандартную инициализацию. Конечно же, инициализировать объект "настоящими" значениями все равно придется в какой-то момент. Поэтому всегда для класса создаются свои конструкторы, синтаксически являющиеся методами, - их может быть несколько. В классе " `Личность` " таким конструктором с параметрами является метод `InitPerson`. Заметьте, мы могли бы определить еще один конструктор, который в отличие от первого, проводил бы полную инициализацию всех свойств, включая Отчество нашей личности.

Деструктор вызывается автоматически при уничтожении объекта. В VBA нет динамического уничтожения объекта в момент, определенный программистом, объекты уничтожаются также как и переменные, при выходе из области их действия. Поэтому деструктор, как правило, не пишется.

Стандартные события Initialize и Terminate

Говоря о конструкторах и деструкторах, необходимо выделить следующее. Роль конструктора по умолчанию в классах VBA играет обработчик события **`Initialize`**. Это общее для многих объектов событие, встречается, когда объект загружается, для объектов - экземпляров классов оно возникает при создании объекта. У обработчика этого события нет параметров, поэтому он играет роль конструктора по умолчанию, не имеющего параметров. Роль деструктора играет обработчик события **`Terminate`**. Он вызывается, когда все ранее установленные ссылки на экземпляр объекта получают

значение **Nothing** или все указатели перестают существовать, выйдя из области своего определения. Заметьте, что при ненормальном завершении программы это событие не возникает. Обработчик этого события (деструктор) пишется значительно реже, поскольку в момент его вызова объект и так корректно будет уничтожен.

Два конструктора класса Rational

Вернемся теперь к нашему классу и определим для него конструктор по умолчанию и "настоящий" свой конструктор:

```
'Конструкторы класса Rational
Private Sub Class_Initialize()
    'Конструктор по умолчанию
    'инициализирует рациональное число дробью 1/1
    m = 1
    n = 1
End Sub

Public Sub CreateRational(ByVal a As Integer, ByVal b As Integer)
    'Собственный конструктор
    'Выполняет довольно сложные действия,
    'прежде чем свойства получат значения
    Dim d As Integer    'Наибольший общий делитель a и b
    If b = 0 Then
        MsgBox " Ошибка при создании рационального числа!" _
        & Chr(13) & "Знаменатель не должен равняться 0."
    Else
        ' приведение знака
        If b < 0 Then
            b = -b: a = -a
        End If
        ' приведение к несократимой дроби
        d = nod(a, b)       ' d - НОД(a,b)
        m = a \ d
        n = b \ d
    End If
End Sub

' Скрытая функция вычисления НОД(m,n)
Private Function nod(ByVal m As Integer, ByVal n As Integer) As
Integer
    Dim p As Integer
    m = Abs(m): n = Abs(n)
    If n > m Then
        p = m: m = n: n = p
    End If
    Do
```

```
        p = m Mod n: m = n: n = p
    Loop Until n = 0
    nod = m
End Function
```

Конструктор по умолчанию, являясь обработчиком события `Initialize`, строился, как обычный обработчик события, используя стандартную заготовку. Он, естественно, является закрытым, поскольку вызывается только системой и не доступен для обычного вызова. Наш собственный обработчик открыт. Он выполняет довольно сложную работу и не сводится к простому присвоению свойств объекта. Поэтому пришлось написать вспомогательную функцию вычисления наибольшего общего делителя. Поскольку она носит служебный характер, то она закрыта и не доступна для вызова при работе с объектами класса `Rational`.

Процедуры - свойства

Для того чтобы можно было получить доступ к закрытым свойствам, предусмотрены специальные процедуры - свойства:

- `Property Let` позволяет установить новое значение терминального свойства, выполняя операцию `Write` (присваивание).
- `Property Set` выполняет те же действия, что и предыдущая процедура, но применима к свойствам - участникам. Вы понимаете, что в VBA присваивание значений обычным переменным и объектам выполняется двумя различными операторами -`Let` и `Set`.
- `Property Get` является дополнительной к предыдущим процедурам, выполняя операцию чтения `Read`. Она применима к терминальным свойствам и свойствам - участникам.

Рассмотрим пять стратегий применения свойств. В первой стратегии `Read - Write` каждому закрытому свойству будет соответствовать пара взаимно дополняющих процедур - свойств `Let ( Set ) - Get`. Эту стратегию мы реализовали для свойств класса " `Личность` ". Стратегия `Write-once` предполагает, что значение свойства должно быть записано только при первом обращении, а далее не должно изменяться. Для реализации этой стратегии процедуры - свойства `Let` и `Set` должны включать в себя код, осуществляющий специальную проверку, проводилось ли ранее присвоение значения свойству. Для объектов это проверка типа " `value Is Nothing` ", для переменных типа `Variant` - " `value = Empty` " и так далее. Мы не станем приводить подробный пример, подобную проверку мы осуществляли при работе со свойством `Отчество` в процедуре ***`WhoIs`***класса " `Личность`. В следующих двух стратегиях используется только одна из процедур пары, либо `Get`, либо `Let ( Set )`. Пятая стратегия предполагает, что доступ к свойствам закрыт, и процедуры - свойства `Get`, `Let ( Set )` в классе вообще не объявляются.

Если свойство имеет тип `Variant`, его значениями могут быть как обычные данные (числа, строки), так и объекты. В этой ситуации необходимо реализовать все три процедуры, применяя `Let` или `Set` в зависимости от того, с чем мы имеем дело - объектом или переменной.

Создание процедуры- свойства

В классе `Rational` есть два закрытых свойства, заданных переменными m и n и определяющими соответственно числитель и знаменатель дроби, представляющей рациональное число. Зададим для каждого из этих свойств пару `Property Let` -

Property Get с именами Числитель и Знаменатель соответственно. Для создания процедур - свойств обычной практикой является использование заготовок, создаваемых автоматически. Выбрав в Редакторе Visual Basic пункт меню Insert | Procedure, мы задали в появившемся диалоговом окне Add Procedure значение Property для типа процедуры. Задав еще имя процедуры - свойства "Числитель ", и, щелкнув OK, мы получили две стандартные заготовки Property Let и Property Get.

Рис. 1. Создание процедур - свойств

Повторив эти действия, мы получили вторую пару заготовок с именем " Знаменатель ". Заготовки затем наполняются, как обычно, содержанием и слегка модифицируются. В заготовках предусмотрен общий тип Variant для параметра в Property Let и возвращаемого значения в Property Get. Разумно изменить этот общий тип на конкретный тип, используемый в данной ситуации. Естественно, что иногда приходится заменить Let на Set или руками добавить еще одну заготовку для Set. Приведем тексты этих процедур для класса Rational после их модификации и заполнения:

```
Public Property Get Числитель() As Integer
  Числитель = m
End Property

Public Property Let Числитель(ByVal NewValue As Integer)
   CreateRational NewValue, n
End Property

Public Property Get Знаменатель() As Integer
  Знаменатель = n
End Property

Public Property Let Знаменатель(ByVal NewValue As Integer)
   CreateRational m, NewValue
End Property
```

Для задания новых значений свойств в процедурах `Let Числитель` и `Let Знаменатель` мы вызываем конструктор, который может изменить нужным образом и числитель и знаменатель.

Замечание: Для класса `Rational` доступ к свойствам `Числитель` и `Знаменатель` следовало бы закрыть полностью. Рациональные числа представляются для пользователей неделимыми и их внутренняя структура должна быть недоступной. Для них следовало бы использовать пятую стратегию работы со свойствами, когда свойства `Let` и `Get` не объявляются.

Процедуры - свойства `Property Let`, `Property Set`, `Property Get` играют важную роль в определении большинства классов. Поэтому имеет смысл разобрать их точный синтаксис.

Синтаксис Let, Get и Set

Оператор Property Let используется для установки значения терминального свойства и имеет следующий синтаксис:

```
[Public | Private | Friend] [Static] Property Let имя-свойства
          ([список-параметров,] значение)
          [Операторы]
          [Exit Property]
          [Операторы]
End Property
```

Оператор Property Set используется для установки значения свойства - участника (объекта) и имеет следующий синтаксис:

```
[Public | Private | Friend] [Static] Property Set имя-свойства
          ([список-параметров,] ссылка)
          [Операторы]
          [Exit Property]
          [Операторы]
End Property
```

Оператор Property Get используется для получения значения свойства и имеет следующий синтаксис:

```
[Public | Private | Friend] [Static] Property Get имя-свойства
           [(список-параметров)] [As Type]
           [Операторы]
           [Exit Property]
           [Операторы]
           [имя-свойства = выражение]
End Property
```

Рассмотрим детали синтаксиса:

- Ключевое слово Public означает доступность Property - свойств во всех процедурах во всех модулях во всех проектах, если только нет дополнительных ограничений. Ключевое слово Private хотя и возможно синтаксически, но лишено смысла, -Property пишут для того, чтобы они были открытыми.
- Ключевое слово Static, как обычно, означает, что значения локальных переменных процедуры, если они есть, не будут изменяться в промежутке между ее вызовами.

- Аргумент имя-свойства задает имя определяемого и изменяемого свойства. Заметьте, что когда задается пара процедурProperty Let (Set) - Get или все три процедуры, все они имеют одно и тоже имя.
- Необязательный список-параметров используется чаще всего при задании свойства, значения которого образуют массив. В нем через запятую перечисляются параметры, передаваемые процедуре, например, в качестве индексов. Парные процедуры должны иметь один и тот же список параметров. Синтаксис списка-параметров такой же, как и у параметров обычных процедур, определяемых оператором Sub. Как обычно, если нужно передать переменное число параметров, то можно использовать ParamArray.
- Параметр значение в Property Let и ссылка в Property Set - это имя переменной (объекта), значение которого передается свойству. Тип значения, возвращаемого процедурой Property Get, должен совпадать с соответствующим типом параметра значение (ссылки).
- Последовательность операторов операторы задает программу вычисления значения свойства. В теле процедуры можно использовать оператор Exit Property для немедленного выхода из процедуры.

Классы, как шаблоны

В этом разделе пойдет речь об одном из их довольно неожиданных применений, когда они позволяют выступать классу в роли контейнера для внутренних и служебных функций.

Построим класс Plan, в котором есть только одно свойство, хранящее текущий месяц. Хранится это свойство, как целое, но для внешнего мира оно выглядит как обычное имя месяца. Вот как это реализовано.

```
'Класс Plan
'Свойство класса
Private CurMonth As Integer
'Закрытый массив, играющий служебную роль
Private Месяцы(1 To 12) As String

Private Sub Class_Initialize()
                        CurMonth = Month(Now)

                        Месяцы(1) = "Январь": Месяцы(2) =
"Февраль": Месяцы(3) = "Март"
                        Месяцы(4) = "Апрель": Месяцы(5) = "Май":
Месяцы(6) = "Июнь"
                        Месяцы(7) = "Июль": Месяцы(8) =
"Август": Месяцы(9) = "Сентябрь"
                        Месяцы(10) = "Октябрь": Месяцы(11) =
"Ноябрь": Месяцы(12) = "Декабрь"
End Sub

Public Property Get ТекущийМесяц() As String
  ТекущийМесяц = Месяцы(CurMonth)
End Property

Public Property Let ТекущийМесяц(ByVal NewValue As String)
Dim i As Byte
```

```
i = 1
Do While i <= 12
  If Месяцы(i) = NewValue Then
    CurMonth = i
    Exit Do
  End If
  i = i + 1
 Loop
If i = 13 Then 'Неверно задан месяц
  CurMonth = Month(Now)
End If
End Property
```

Здесь работу с закрытым свойством CurMonth обеспечивают процедуры - свойства Property Let ТекущийМесяц иProperty Get ТекущийМесяц. С их помощью можно читать и изменять значение свойства CurMonth. Одновременно Let и Getзанимаются преобразованием данных, что является распространенной практикой. Пример этот интересен и с позиций построения конструктора по умолчанию, который выполняет все необходимые внутренние инициализации, в данном случае, задавая значение массиву Месяцы.

Приведем еще процедуру, которая работает с данными этого класса:

```
Public Sub WorkWithPlan()
 Dim MyPlan As New Plan
 Debug.Print MyPlan.ТекущийМесяц
 MyPlan.ТекущийМесяц = "Апрель"
 Debug.Print MyPlan.ТекущийМесяц
 MyPlan.ТекущийМесяц = "Янв."
 Debug.Print MyPlan.ТекущийМесяц


End Sub
```

В результате ее работы, (процедура запускалась в марте месяце) напечатаны значения:

```
Март
Апрель
Март
```

При обращении к свойству класса необхлдимо получать значение текущего месяца. Но ведь текущий месяц не должен изменять пользователь. Так что следовало бы воспользоваться стратегией Read- only и не разрешать изменять значение свойства. Поэтому исключим Property Let из нашего класса. Но в этом примере есть куда более интересная особенность. Исключив Property Let, мы понимаем, что теперь теряет смысл хранение самого свойства CurMonth. Его значение мы можем (и должны по смыслу свойства) получать динамически, обращаясь к системной процедуре Month(Now). Удалим из класса и переменную CurMonth. Наш новый класс PlanNew стал проще:

```
'Класс PlanNew
'Закрытый массив, играющий служебную роль
```

```
Private Месяцы(1 To 12) As String

Private Sub Class_Initialize()
Месяцы(1) = "Январь": Месяцы(2) = "Февраль": Месяцы(3) = "Март"
Месяцы(4) = "Апрель": Месяцы(5) = "Май": Месяцы(6) = "Июнь"
Месяцы(7) = "Июль": Месяцы(8) = "Август": Месяцы(9) = "Сентябрь"
Месяцы(10) = "Октябрь": Месяцы(11) = "Ноябрь": Месяцы(12) =
"Декабрь"
End Sub

Public Property Get ТекущийМесяц() As String
  ТекущийМесяц = Месяцы(Month(Now))
End Property
```

Свойство ТекущийМесяц теперь имеет статус Read- only. Но главная суть не в этом. Теперь отчетливо видна настоящая роль класса PlanNew, - он является контейнером для вызова служебных функций Month(Now).

Для одной функции, как в нашем примере, вероятно, не стоит создавать класс, но для десятка функций это вполне целесообразно. Эта техника с особым успехом используется, когда нужно упаковать обращение к Win API функциям. Так что, подводя итог, отметим, что иногда классы используются как упаковка, расширяющая стандартные возможности и особую роль в этом процессе играют процедуры - свойства.

Семейство классов и процедуры - свойства

Когда создается семейство классов, без свойств - участников не обойтись. В классах семейства, основанных на механизме встраивания, обязательно есть свойства, представляющие собой объекты. Эти свойства, как и терминальные, обычно, закрываются и для работы с ними используются процедуры - свойства Property Set и Property Get. Иногда приходится передавать и дополнительные параметры при работе с ними. Рассмотрим типичный пример. Ранее мы построили класс " Личность ", теперь построим класс " Группа ", содержащий группу личностей. При построении этого класса ограничимся минимальными средствами, необходимыми для демонстрации работы со свойствами:

```
'Класс Группа Личностей
Const РазмерГруппы As Byte = 25
'Свойства
Private Group(1 To РазмерГруппы) As Личность

'Процедуры-свойства
Public Property Get ЧленГруппы(num As Byte) As Личность
  'Если номер корректен
  If (num >= 1) And (num <= РазмерГруппы) Then
    'Если существует в группе личность с таким номером
    If Not (Group(num) Is Nothing) Then
      Set ЧленГруппы = Group(num)
    Else: MsgBox ("В группе нет человека с номером " & num)
    End If
```

```
  Else: MsgBox ("Некорректно задан номер в группе - " & num)
  End If
End Property

Public Property Set ЧленГруппы(num As Byte, NewValue As Личность)
 'Если номер корректен
  If (num >= 1) And (num <= РазмерГруппы) Then
    'Если в группе нет личности с таким номером, то она создается
    If Group(num) Is Nothing Then
      Set Group(num) = NewValue
    Else: MsgBox ("В группе уже есть человек с номером " & num)
    End If
  Else: MsgBox ("Некорректно задан номер в группе - " & num)
  End If
End Property

Public Sub Сведения()
  Dim i As Byte
  For i = 1 To РазмерГруппы
    If Not (Group(i) Is Nothing) Then
      Group(i).PrintPerson
    End If
  Next i
End Sub
```

Массив объектов класса " Личность " является закрытым свойством класса " Группа ". Процедуры - свойства Property Get ЧленГруппы и Property Set ЧленГруппы обеспечивают доступ к элементам этого массива. Индекс элемента является дополнительным параметром. Процедура Set имеет статус Write-once, - если элемент с заданным номером уже определен в группе, то он не переопределяется. При создании этих процедур, нам, конечно же, пришлось модифицировать стандартные заготовки Let и Get. Для полноты картины приведем процедуру, в которой показано, как работать с группой:

```
Public Sub WorkWithGroup()
                          Dim UserOne As New Личность
  Dim UserTwo As New Личность
  Dim UserThree As New Личность
  Dim Знакомые As New Группа
 Dim NewUser As New Личность
  'Личности
  UserOne.InitPerson FN:="Петр", LN:="Петров", DoB:=#1/23/1968#
  UserTwo.InitPerson FN:="Анна", LN:="Козлова", DoB:=#7/21/1968#
  UserThree.InitPerson FN:="Анна", LN:="Агеева", DoB:=#5/17/1803#
  'Группа
  Set Знакомые.ЧленГруппы(1) = UserOne
  Set Знакомые.ЧленГруппы(2) = UserTwo
```

```
  Set Знакомые.ЧленГруппы(1) = UserThree
  Set Знакомые.ЧленГруппы(3) = UserThree
  Set NewUser = Знакомые.ЧленГруппы(7)
  Set NewUser = Знакомые.ЧленГруппы(3)
  Знакомые.Сведения

End Sub
```

Без процедур - свойств можно обойтись, заменив их обычными методами класса. Правда, такая замена приведет, обычно, к некоторой потере эффективности. Но методы являются основным способом работы с данными (свойствами), определяя поведение объектов класса.

Объекты, используемые в Visual Basic, берутся из внутренних и внешних источников. Примерами внутренних объектов являются встроенные объекты и классы проекта. Примерами внешних объектов являются сборки и объекты COM.

Внутренние объекты

Внутренние (или *встроенные*) объекты — это объекты, которые изначально присутствуют в Visual Basic. К ним относятся входят простые скалярные типы, например **Integer** и **Double**, а также **Array** и **String**. Нет необходимости создавать ссылки на внутренние объекты перед их использованием в проекте.

Другими внутренними объектами являются экземпляры классов в текущем проекте. Эти классы используются повсюду в пределах проекта, и их можно сделать доступными для использования другими приложениями при создании сборки.

Внешние объекты

Внешние объекты — это объекты, находящиеся в других проектах или сборках, которые недоступны вашему проекту по умолчанию. Перед использованием внешних объектов в проекте на них нужно создать ссылки.

Объект представляет экземпляр класса, например Form или Label. Перед получением доступа к членам, не являющимся общими, необходимо создать объект. Чтобы сделать это, используйте зарезервированное слово New, чтобы указать класс, из которого следует создать объект, и сохраните новый объект в переменной объекта.

```
Dim warningLabel As New System.Windows.Forms.Label
```

Элементы экземпляра и общие члены

При создании объекта из класса результатом является экземпляр этого класса. Члены, не объявленные с ключевым словом Shared , являются *членами экземпляра*, которые принадлежат исключительно определенному экземпляру. Член экземпляра в одном экземпляре не зависит от того же члена в другом экземпляре того же класса. Например, переменная члена экземпляра может иметь различные значения в различных экземплярах.

Члены, объявляемые с помощью зарезервированного слова **Shared** являются *общими членами*, которые относятся к классу в целом, а не к любому определенному экземпляру. Общий элемент существует только один, независимо от количества созданных экземпляров класса, или даже в том случае, если не создавать экземпляры. Например, переменная общего члена имеет только одно значение, которое доступно всем кодам, имеющим доступ к классу.

Intellisense список элементов

Технология IntelliSense перечисляет члены класса при вызове параметра List Members, например при вводе точки (.) в качестве оператора доступа к членам. Если ставится точка после имени переменной, объявленной как экземпляр этого класса, то IntelliSense перечисляет все члены экземпляра и ни один из общих членов. Если ставится точка перед именем самого класса, то IntelliSense перечисляет все общие члены и ни один из членов экземпляров.

Поля и свойства

Хранящаяся в объекте информация предоставляется *полями* и *свойствами*. Можно получить и задать их значения с помощью инструкций присваивания так же, как получить и задать локальные переменные в процедуре. В следующем примере извлекается свойство Width и устанавливается свойство ForeColor объекта Label.

```
Dim warningWidth As Integer = warningLabel.Width
warningLabel.ForeColor = System.Drawing.Color.Red
```

Доступ к членам.

После создания объекта можно обращаться к его полям, свойствам, методам и событиям с помощью объектной переменной. Если член является Shared , для доступа к нему объект создавать необязательно.

Для доступа к члену объекта нужно указать последовательно имя объекта, точку (.) и имя необходимого члена. В следующем примере устанавливается свойствоText объекта Label.

```
warningLabel.Text = "Data not saved"
```

Передача объектов в процедуры

Visual Basic позволяет передавать объекты как аргументы для процедур так же, как передаются другие типы аргументов. Это показано в следующих примерах.

Передача нового экземпляра формы процедуре

```
Private Sub Button1_Click(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles Button1.Click

  Dim newForm As New Form1
  newForm.Show()
  CenterForm(newForm)
End Sub

Sub CenterForm(ByVal TheForm As Form)
  ' Centers the form on the screen.
  Dim RecForm As Rectangle = Screen.GetBounds(TheForm)
  TheForm.Left = CInt((RecForm.Width - TheForm.Width) / 2)
  TheForm.Top = CInt((RecForm.Height - TheForm.Height) / 2)
End Sub
```

Установка и получение значений свойств

При работе с формами и элементами управления в Visual Basic можно установить их свойства программно во время выполнения или в режиме разработки с помощью окна Свойства. Свойства большинства других объектов (например, объектов из сборок или объектов, созданных пользователем) устанавливаются только программно.

Свойства, доступные для установки и чтения, называются *свойствами, доступными для чтения и записи*. Свойства, доступные для чтения, но не для изменения, называются *свойствами только для чтения*. Свойства, доступные для записи, но не для чтения, называются *свойствами только для записи*.

Значение свойства устанавливается при необходимости изменения внешнего вида или поведения объекта. Например, чтобы изменить содержимое текстового поля, следует изменить свойство `Text` элемента управления "Текстовое поле".

Значение свойства извлекается, когда необходимо выяснить состояние объекта до выполнения кодом некоторых действий, таких как присвоение значения другому объекту. Например, свойство `Text` элемента управления "Текстовое поле" возвращают для определения содержимого текстового поля перед запуском кода, который может изменить значение.

- Задание значения свойству

Воспользуйтесь следующим синтаксисом.

Объект.свойство=выражение

Следующие операторы предоставляют примеры установки свойств.

```
' Set the Top property to 200 twips.
TextBox1.Top = 200
' Display the text box.
TextBox1.Visible = True
' Display 'hello' in the text box.
TextBox1.Text = "hello"
```

- Получение значений свойств

Воспользуйтесь следующим синтаксисом.

переменная=объект.свойство

Кроме того, можно получить значение свойства в виде части более сложного выражения без присвоения значения свойства переменной. Следующий код изменяет свойство **Top** элемента управления "Переключатель".

```
RadioButton1.Top += 20
```

МЕТОДЫ

Действие, которое выполняет объект, называется *методом*. Например, Add является методом объекта ComboBox, он добавляет новую запись в поле со списком.

В данном примере демонстрируется метод Start класса Timer.

```
Dim safetyTimer As New System.Windows.Forms.Timer
safetyTimer.Start()
```

Выполнение действий с помощью методов

Методы — это процедуры, связанные с объектами. В отличие от полей и свойств, представляющих собой сведения, которые объект способен сохранить, методы — это действия, которые объект способен выполнить. Методы могут влиять на значение свойств. Например, используя аналогию с радио, можно использовать метод `SetVolume`, чтобы изменить значение свойства `Volume`. Аналогично в Visual Basic элементы списка имеют свойство `List`, которое можно изменить с помощью методов `Clear` и `Add`.

При использовании метода в коде способ записи оператора зависит от количества аргументов, необходимых методу, и от того, возвращает ли он значение. Как правило, методы используются точно так же, как вызовы подпрограмм или функций. Если говорить более точно, методы вызываются так же, как модульные процедуры, за исключением того, что название метода предваряется с помощью выражения, указывающего на экземпляр объекта, метод которого необходимо вызвать. Если имя объекта не указано, экземпляр неявно является переменной **Me**.

Чтобы использовать метод, для которого не требуются аргументы

Воспользуйтесь следующим синтаксисом.

Объект.метод()

В следующем примере с помощью метода `Refresh` перерисовывается графическое окно.

```
' Force the control to repaint.
PictureBox1.Refresh()
```

Примечание. Некоторые методы, например Refresh, не имеют аргументов и не возвращают значений

Использование метода, для которого требуется несколько аргументов

Поместите аргументы в круглые скобки и разделите их запятыми. В следующем примере метод **MsgBox** использует аргументы, которые указывают сообщение для отображения и стиль окна сообщения.

```
MsgBox("Database update complete", _
  MsgBoxStyle.OKOnly Or MsgBoxStyle.Exclamation, _
  "My Application")
```

Чтобы использовать метод, который возвращает значение

Назначьте возвращаемое значение переменной или непосредственно воспользуйтесь вызовом метода как параметром для другого вызова. Следующий код сохраняет возвращаемое значение:

```
Dim Response As MsgBoxResult
Response = MsgBox("Do you want to exit?", _
     MsgBoxStyle.YesNo Or MsgBoxStyle.Question, _
     "My Application")
```

В следующем примере используется значение, возвращаемое методом **Len** в виде аргумента для **MsgBox**.

```
Dim TestStr As String = "Some String"
' Display the string "String length is : 11".
MsgBox("String length is : " & Len(TestStr))
```

Отношения между объектами

Связь объектов друг с другом может быть различной. Основными видами связи являются *hierarchical* и *containment*.

- **Иерархическая связь.**

При *иерархической связи(hierarchical relationship)* одни классы являются наследниками других, более фундаментальных классов. Иерархии классов удобны при описании объектов — подтипов более универсальных классов. Например, в пространстве имен System.Windows.Forms классы Label и TextBox являются производными от класса Control. Производные классы наследуют члены класса, от которого они произведены, позволяя добавлять функциональность по мере продвижения вглубь иерархии классов.

- **Отношение вложенности**

Другим способом описания отношений между объектами является *containment relationship*. Объекты-контейнеры инкапсулируют другие объекты. Например, объект OperatingSystem логически содержит объект Version, который возвращается с помощью его свойства Version. Обратите внимание, что объект контейнера физически не содержит любые другие объекты.

Коллекции

Один из типов отношения вложенности представлен *collections*. Коллекции представляют собой группы подобных объектов, которые могут быть перечислены.Visual Basic поддерживает определенный синтаксис в Инструкция For Each... Next , позволяющем перебирать элементы коллекции. Кроме того, коллекции часто позволяют использовать свойство Свойство Item (объект Collection) для извлечения элементов по индексу или привязки их к уникальной строке. Коллекции более просты в применении, чем массивы, т.к. они позволяют добавлять или удалять элементы без использования индексов. Коллекции просты в обращении и потому часто используются для хранения форм и элементов управления.

Наследование класса в Visual Basic

В этом примере определяются классы `Circle` и `Rectangle`, которые оба наследуются от класса `Shape`, и класс `Square`, наследуемый от класса `Rectangle`.

```
Public Class Shape
 ' Definitions of properties, methods, fields, and events.
End Class
Public Class Circle : Inherits Shape
 ' Specialized properties, methods, fields, events for Circle.
End Class
Public Class Rectangle : Inherits Shape
 ' Specialized properties, methods, fields, events for Rectangle.
End Class
Public Class Square : Inherits Rectangle
 ' Specialized properties, methods, fields, events for Square.
End Class
```

Примечание. Убедитесь, что класс, от которого вы хотите наследовать, не определен как **NotInheritable**.

В следующем примере реализован механизм наследования при создании класса.

Задача. Нарисовать на форме рисуются контуры эллипса и прямоугольника.

```
Private Sub DrawEllipse()
 Dim myPen As New System.Drawing.Pen(System.Drawing.Color.Red)
 Dim formGraphics As System.Drawing.Graphics
 formGraphics = Me.CreateGraphics()
 formGraphics.DrawEllipse(myPen, New Rectangle(0, 0, 200, 300))
 myPen.Dispose()
 formGraphics.Dispose()
End Sub

Private Sub DrawRectangle()
 Dim myPen As New System.Drawing.Pen(System.Drawing.Color.Red)
 Dim formGraphics As System.Drawing.Graphics
 formGraphics = Me.CreateGraphics()
 formGraphics.DrawRectangle(myPen, New Rectangle(0, 0, 200, 300))
 myPen.Dispose()
 formGraphics.Dispose()
End Sub
```

ИСПОЛЬЗОВАНИЕ ЯЗЫКА VBA ДЛЯ РАЗРАБОТКИ ПРИЛОЖЕНИЙ

Язык Basic в его базовом варианте - мягко говоря, не такое уж мощное средство. Однако, облаченный в оболочку (IDE), с API через плечо, с ActiveX-совместимостью - через другое и достаточно закрытый для посредственного круга пользователей, позволяет смело соперничать с Delphi (из той же оперы: сначала повзрослел Паскаль до статуса Объектного, затем - признание), Visual C++ - молниеносно создавать сложнейшие программы для Windows. Runtime-файлы msvbvm*0.dll содержат стандартный ассортимент для UI, - в виде экземпляров классов, присутствующих сейчас в системе. Вместе с тем, он намного проще в реализации пользовательского интерфейса, нежели у программиста на C++. Кроме того, скорость в работе Си-программ почти не отличается от скорости современного (v.5, 6) Бейсик-софта (для того, чтобы заметить разницу, необходимо писать специальные счетчики-секундомеры)..

Внутренние объекты VBA

Каждый объект имеет свои собственные специфические качества и поведение. Объектами можно манипулировать, задавать и изменять их свойства и вызывать их методы. Если считать объект существительным, то свойства объекта - это прилагательные, а методы - глаголы. Хорошим инструментом для изучения основ объектно-ориентированного программирования является VBA, имеющий встроенные объекты. Объекты VBA имеют свои свойства и методы. Объектно-ориентированное приложение организует данные и выполняемые операторы программного кода в связанные объекты, что облегчает разработку, организацию и работу со сложными структурами данных и действиями, выполняемыми над этими над этими данными. Каждый объект в программном приложении содержит данные и операторы, связанные вместе и образующие единый элемент. Большинство приложений содержат много различных типов объектов. В зависимости от того, в какое приложение встроен язык VBA, варьируются и объекты, с которыми он работает. Объекты программирования организуются в виде иерархии, которая называется объектной моделью приложения.

В Excel, например, объектами VBA являются

- приложение Excel (Application),
- книги (Workbook),
- листы (Worksheet),
- диапазоны данных (Range),
- диаграммы (Chart),
- диалоговые окна (Window).

Объекты VBA в Word – документы (Document), диапазоны текста (Range), таблицы (Table), графические объекты, диалоговые окна и само приложение Word.

Все host-приложения VBA, такие как Outlook, Access и Microsoft Project, имеют объекты, доступные для VBA таким же образом как объекты Word и Excel. Конкретные объекты в host-приложении VBA варьируются в зависимости от приложения. Объекты в Access, например, все имеют отношение к базам данных и манипулированию базами данных, в то время как объекты в Excel связаны с понятиями рабочих листов, книг и так далее. Просмотреть объекты VBA для того или иного приложения можно в редакторе Visual Basic, нажав клавишу F2 на клавиатуре либо кнопку ”Object Browser” в стандартной панели инструментов (рис. 1,2).

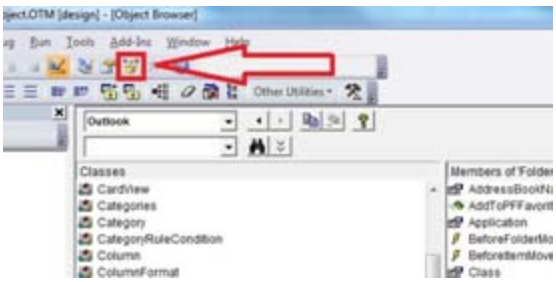

Рис. 1. Object Browser

VBA представляет средства для использования имеющихся и создания новых объектов. Класс — это общее описание однородных по структуре объектов. Класс задает характеристики и поведение объектов. Классы в VBA оформляются в виде отдельных модулей.

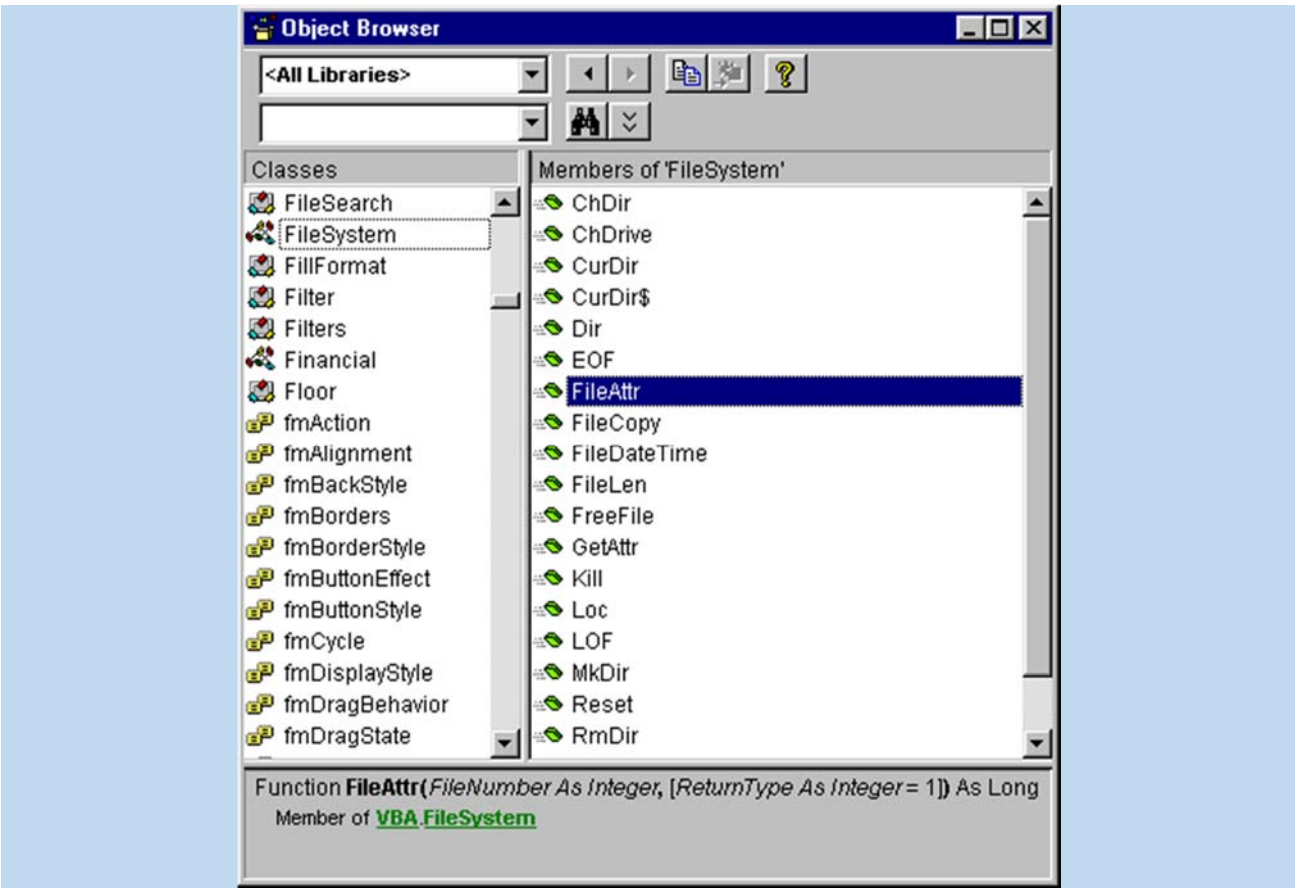

Рис.2. Список объектов VBA

Метод InputBox и функция InputBox объекта Application

Метод *InputBox* объекта *Application* имеет три преимущества по сравнению с функцией *InputBox*:

- возможность задать тип возвращаемого значения;
- возможность указать диапазон рабочего листа;
- автоматическая проверка правильности введённых данных.

Метод *InputBox* имеет следующий синтаксис:

InputBox(*Prompt* [, *Title*] [, *Default*] [, *Left*] [, *Top*] [, *Helpfile*, *Context*] [, *Type*])

Метод *InputBox* имеет те же параметры, что и функция *InputBox*, и один дополнительный параметр *Type*, который указывает, данные какого типа должен возвращать метод. Этот параметр является необязательным, и в случае его отсутствия метод возвращает строку. Если введённое значение не соответствует требуемому типу, приложение Microsoft Excel выдаёт сообщение об ошибке и позволяет пользователю ввести другое значение.

Код	Тип возвращаемого значения
0	Формула
1	Число
2	Строка
4	Логическое значение
8	Диапазон
16	Ошибка

```
Range("F2").value = Application.InputBox("Введите число", "Ввод числа", Type:=1)
Range("F3").value = Application.InputBox("Введите строку", "Ввод строки", Type:=2)
Range("F4").value = Application.InputBox("Введите логическое значение", _
      "Ввод логического значения", Type:=4)
Range("F1").value = Application.InputBox("Введите формулу", "Ввод формулы", Type:=0)

Sub EraseRange()
 Dim userRange As Range

 On Error GoTo Canceled
 Set userRange = Application.InputBox(Prompt:="Удаляемый диапазон:", _
       Title:="Удаление диапазона", _
       Default:=Selection.Address, Type:=8)
 userRange.Clear
 userRange.Select
Canceled:
End Sub
```

Функция VBA MsgBox

Функция VBA *MsgBox* предоставляет пользователю простой способ отображения сообщения. Также она возвращает реакцию пользователя на запрос. Функция *MsgBox* имеет следующий синтаксис:

MsgBox(*Prompt* [, *Buttons*] [, *Title*] [, *Helpfile*, *Context*])

Параметр *Buttons* определяет, какие кнопки и значки будут отображены в окне.

Константа	Значение	Назначение
vbOKOnly	0	Отображается только кнопка *ОК* (по умолчанию)
vbOKCancle	1	Отображаются кнопки *ОК* и *Отмена*
vbAbortRetryIgnore	2	Отображаются кнопки *Прервать*, *Повтор* и *Пропустить*
vbYesNoCancel	3	Отображаются кнопки *Да*, *Нет* и *Отмена*
vbYesNo	4	Отображаются кнопки *Да* и *Нет*
vbRetryCancel	5	Отображаются кнопки *Повтор* и *Отмена*
vbCritical	16	Отображается значок важного сообщения
vbQuestion	32	Отображается значок запроса
vbExclamation	48	Отображается значок предупреждающего сообщения
vbInformation	64	Отображается значок информационного сообщения
vbDefaultButton1	0	По умолчанию выделена первая кнопка
vbDefaultButton2	256	По умолчанию выделена вторая кнопка
vbDefaultButton3	512	По умолчанию выделена третья кнопка

Функция *MsgBox* возвращает число, соответствующее кнопке, нажатой пользователем.

Константа	Значение	Нажатая кнопка
vbOK	1	ОК
vbCancel	2	Отмена
vbAbort	3	Прервать
vbRetry	4	Повтор
vbIgnore	5	Пропустить
vbYes	6	Да
vbNo	7	Нет

```
MsgBox("Сообщение", vbOKOnly + vbInformation, "Заголовок")

ans = MsgBox("Сообщение", vbOKCancel + vbQuestion, "Заголовок")
...
ans = MsgBox("Сообщение", vbRetryCancel + vbCritical, "Заголовок")
...
if MsgBox("Сообщение", vbYesNo + vbExclamation + vbDefaultButton2, _
   "Заголовок") = vbYes Then
 ...
Else
 ...
End If
```

Интеллектуальные возможности редактора кода

Написание программ существенно облегчается за счет способности редактора кода автоматически завершать написание операторов, свойств и параметров. При написании кода редактор сам предлагает пользователю список компонентов, логически завершающих вводимую пользователем инструкцию. Например, набирая код

```
Range("A1").
```

после ввода точки на экране отобразится список компонентов (рис. 1.5), которые логически завершают данную инструкцию. Двойной щелчок на выбранном элементе из этого списка или нажатие клавиши <Tab> вставляет выбранное имя в код программы. При этом использование клавиши <Tab> вместо мышки иногда предпочтительней, т. к. эта клавиша находится прямо под рукой и нажатие на нее производится только одним движением пальца левой руки, что не требует особого времени и усилий.

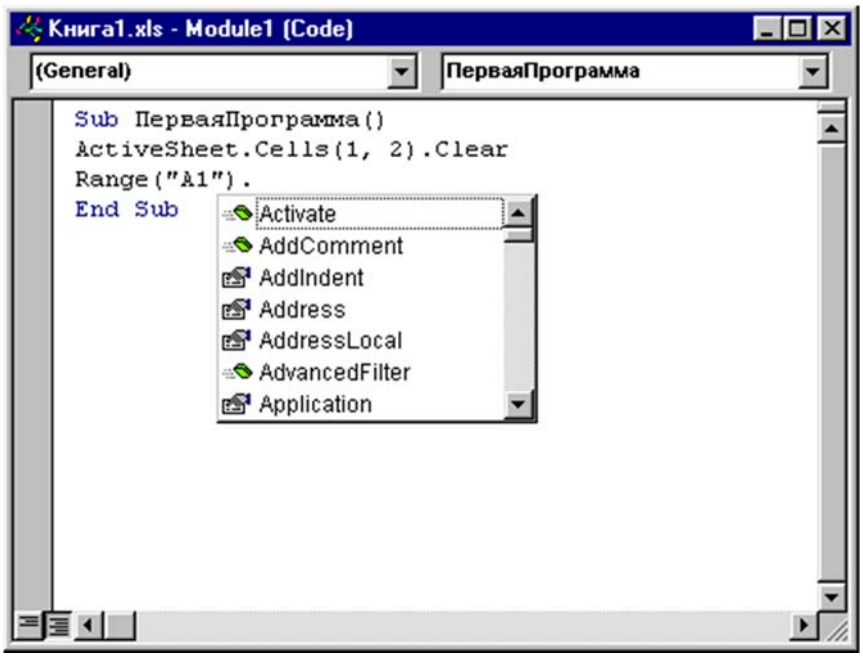

Рис. 3. Список компонентов

Автоматическое отображение списка компонентов происходит только при установленном флажке Список компонентов (Auto List Members) вкладки Редактор (Editor) диалогового окна Параметры (Options), отображаемого на экране после выбора команды Сервис, Параметры (Tools, Options).

Список компонентов можно выводить на экран нажатием комбинации клавиш <Ctrl>+<J>, при этом список отображается как при установленном, так и при снятом флажке Список компонентов (Auto List Members) вкладки Редактор (Editor) диалогового окна Параметры (Options).

Отображение списка компонентов, логически завершающих вводимую инструкцию,, является одним из интеллектуальных качеств редактора кода. Этим качеством интеллектуальные ресурсы редактора кода не исчерпываются. Другим его такого рода качеством является автоматическое отображение на экране сведений о процедурах, функциях, свойствах и методах после набора их имени (рис. 1.6).

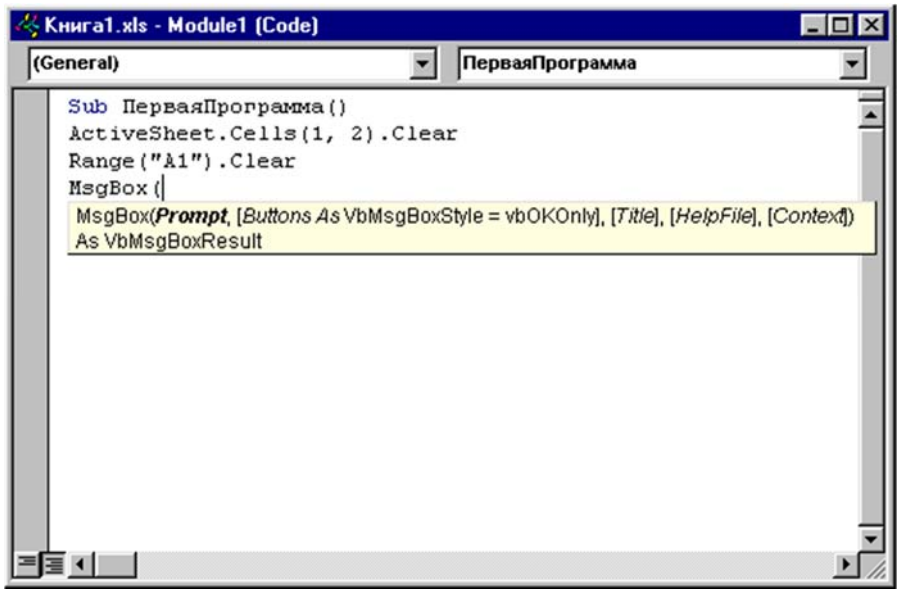

Рис. 4. Отображаемые сведения о вводимой процедуре

Автоматическое отображение на экране сведений о процедурах, функциях, свойствах и методах после ввода их имени происходит только при установленном флажке Краткие сведения (Auto Quick Info) вкладки Редактор (Editor) диалогового окнаПараметры (Options), отображаемого на экране после выбора команды Сервис, Параметры (Tools, Options) (см. рис. 1.4).

Описанную выше всплывающую подсказку можно также выводить на экран нажатием комбинации клавиш <Ctrl>+<!>. При этом всплывающая подсказка отображается как при установленном, так и при снятом флажке «Краткие сведения» вкладки «Редактор» диалогового окна «Параметры»(Options).

Редактор кода также производит автоматическую проверку синтаксиса набранной строки кода сразу после нажатия клавиши <Enter>. Если после набора строки и нажатия клавиши <Enter> строка выделяется красным цветом, то это как раз и указывает на наличие синтаксической ошибки в набранной строке. Эту ошибку необходимо найти и исправить. Кроме того, если установлен флажок Проверка_синтаксиса (Auto Syntax Check) вкладки Редактор (Editor) диалогового окна Параметры (Options), отображаемого на экране посредством выбора команды Сервис, Параметры (Tools, Options) (см. рис. 1.4), помимо выделения красным цветом фрагмента кода с синтаксической ошибкой, на экране отображается диалоговое окно, поясняющее, какая возможная ошибка произошла.

Редактор кода обладает еще одной мощной интеллектуальной возможностью, увеличивающей эффективность работы пользователя. Если курсор расположить на ключевом слове языка VBA, имени процедуры, функции, свойства или метода и нажать клавишу <F1>, то на экране появится окно со справочной информацией об этой функции. Обычно в справке имеется пример использования кода, что позволяет быстрее разобраться в ситуации, которая при написании программы озадачила вас.

Работа с ячейками и диапазонами ячеек. Использование объектов Range и Selection

В Excel наиболее важным является объект ***Application***. Объект Application. Он представляет само приложение Excel и имеет более 120 свойств и 40 методов. Эти свойства и методы предназначены для установки общих параметров приложения Excel.

В иерархии Excel объект ***Workbook*** (рабочая книга) идет сразу после объекта Application и представляет файл рабочей книги.

Однако наиболее «употребляемым» на практике является объект ***Range***, который наилучшим образом отображает возможности использования VBA в Excel (Приложение 2). В иерархии Excel объект Range (диапазон) идет сразу после объекта worksheet. Объект Range является одним из ключевых объектов VBA.

Объект ***selection*** (выделение) возникает в VBA двояко – либо как результат работы метода Select – «выделить», либо при вызове свойства selection. Тип получаемого объекта зависит от типа выделенного объекта. Чаще всего объект Selection принадлежит классу Range, и при работе с ним можно использовать свойства и методы объекта Range. Интересной особенностью объектов Range и Selection является то, что они не являются элементами никакого семейства объектов.

При работе с объектом Range необходимо помнить о неявных и полных ссылках в Excel на ячейку рабочего листа.

Задание групп строк и столбцов с помощью объекта Range

Если в диапазоне указываются только имена столбцов или строк, то объект Range задает диапазон, состоящий из указанных столбцов или строк. Например, Range («a: c») задает диапазон, состоящий из столбцов a, в и c, а Range(«2:2») – из второй строки. Другим способом работы со строками и столбцами являются методы Rows (строки) и columns (столбцы), возвращающие коллекции строк и столбцов. Например, столбцом a является columns (1), а второй строкой – Rows (2).

1.Выбор ячейки на активном листе

Для выбора ячейки D5 на активном листе можно использовать любой из приведенных ниже способов.

```
ActiveSheet.Cells(5, 4).Select
```

-или-

```
ActiveSheet.Range("D5").Select
```

2. Выбор ячейки на другом листе той же книги

Для выбора ячейки E6 на другом листе той же книги можно использовать любой из приведенных ниже способов.

```
Application.Goto ActiveWorkbook.Sheets("Sheet2").Cells(6, 5)
```

-или-

```
Application.Goto (ActiveWorkbook.Sheets("Sheet2").Range("E6"))
```

Кроме того, можно активировать лист и использовать для выбора ячейки способ, описанный в пункте 1.

```
Sheets("Sheet2").Activate
ActiveSheet.Cells(6, 5).Select
```

3. Выбор ячейки на листе другой книги

Для выбора ячейки F7 на листе другой книги можно использовать любой из приведенных ниже способов.

```
Application.Goto Workbooks("BOOK2.XLS").Sheets("Sheet1").Cells(7, 6)
```

-или-

```
Application.GotoWorkbooks("BOOK2.XLS").Sheets("Sheet1").Range("F7")
```

Кроме того, можно активировать лист и использовать для выбора ячейки способ, описанный в пункте 1.

```
Workbooks("BOOK2.XLS").Sheets("Sheet1").Activate
ActiveSheet.Cells(7, 6).Select
```

4. Выбор диапазона ячеек на активном листе

Для выбора диапазона ячеек C2:D10 на активном листе можно использовать любой из приведенных ниже способов.

- ```
  ActiveSheet.Range(Cells(2, 3), Cells(10, 4)).Select
  ```

- ```
  ActiveSheet.Range("C2:D10").Select
  ```

- ```
  ActiveSheet.Range("C2", "D10").Select
  ```

5. Выбор диапазона ячеек на другом листе той же книги

Для выбора диапазона ячеек D3:E11 на другом листе той же книги можно использовать любой из приведенных ниже способов.

```
Application.Goto ActiveWorkbook.Sheets("Sheet3").Range("D3:E11")

Application.Goto ActiveWorkbook.Sheets("Sheet3").Range("D3", "E11")
```

Кроме того, можно активировать лист и использовать для выбора диапазона ячеек способ, описанный в пункте 4.

```
Sheets("Sheet3").Activate
ActiveSheet.Range(Cells(3, 4), Cells(11, 5)).Select
```

6. Выбор диапазона ячеек на листе другой книги

Для выбора диапазона ячеек E4:F12 на листе другой книги можно использовать любой из приведенных ниже способов.

```
Application.Goto Workbooks("BOOK2.XLS").Sheets("Sheet1").Range("E4:F12")

Application.Goto _
  Workbooks("BOOK2.XLS").Sheets("Sheet1").Range("E4", "F12")
```

Кроме того, можно активировать лист и использовать для выбора диапазона ячеек способ, описанный в пункте 4.

```
Workbooks("BOOK2.XLS").Sheets("Sheet1").Activate
 ActiveSheet.Range(Cells(4, 5), Cells(12, 6)).Select
```

7. Выбор ячейки относительно активной ячейки

Для выбора ячейки, расположенной на пять строк ниже и четыре столбца левее активной ячейки, можно использовать приведенный ниже способ.

```
ActiveCell.Offset(5, -4).Select
```

Для выбора ячейки, расположенной на две строки выше и три столбца правее активной ячейки можно использовать приведенный ниже способ.
```

```
ActiveCell.Offset(-2, 3).Select
```

Примечание. При попытке выбрать ячейку за пределами листа произойдет ошибка. Например, при выполнении первого фрагмента ошибка произойдет, если активная ячейка находится в столбцах A-D, так как после перемещения на четыре столбца влево адрес ячейки стал бы неверным.

8. Выбор ячейки относительно другой (неактивной) ячейки

Для выбора ячейки, расположенной на пять строк ниже и четыре столбца правее ячейки C7, можно использовать приведенные ниже способы.

```
ActiveSheet.Cells(7, 3).Offset(5, 4).Select
```

```
ActiveSheet.Range("C7").Offset(5, 4).Select
```

9. Выбор диапазона ячеек относительно указанного диапазона

Для выбора диапазона ячеек, имеющего тот же размер, что и именованный диапазон «Test», но смещенного на четыре строки вниз и три столбца вправо, можно использовать приведенный ниже способ.

```
ActiveSheet.Range("Test").Offset(4, 3).Select
```

Если именованный диапазон расположен на другом (неактивном) листе, сначала активируйте этот лист, а затем выберите диапазон, используя приведенный ниже способ.

```
Sheets("Sheet3").Activate
ActiveSheet.Range("Test").Offset(4, 3).Select
```

10. Выбор указанного диапазона с изменением его размеров

Для выбора именованного диапазона «Database» и его увеличения на пять строк можно использовать приведенный ниже способ.

```
Range("Database").Select
Selection.Resize(Selection.Rows.Count + 5, Selection.Columns.Count).Select
```

11. Выбор указанного диапазона, его смещение и изменение его размеров

Чтобы выбрать диапазон, расположенный на четыре строки ниже и три столбца правее именованного диапазона «Database», и включить в него на две строки и один столбец больше, чем в именованном диапазоне, можно использовать приведенный ниже способ.

```
Range("Database").Select
Selection.Offset(4, 3).Resize(Selection.Rows.Count + 2,Selection.Columns.Count + 1).Select
```

12. Выбор объединения двух или более указанных диапазонов

Для выбора объединения (совмещенной области) двух именованных диапазонов «Test» и «Sample» можно использовать приведенный ниже способ.

```
Application.Union(Range("Test"), Range("Sample")).Select
```

Чтобы этот способ сработал, оба диапазона должны относиться к одному листу. Метод **Union** не поддерживает работу с разными листами. Например, следующая строка дает нужный результат:

```
Set y = Application.Union(Range("Sheet1!A1:B2"), Range("Sheet1!C3:D4"))
```

но при попытке выполнить код

```
Set y = Application.Union(Range("Sheet1!A1:B2"), Range("Sheet2!C3:D4"))
```

будет выведено следующее сообщение об ошибке:

Сбой метода Union класса приложения

13. Выбор пересечения двух или более указанных диапазонов

Для выбора пересечения двух именованных диапазонов «Test» и «Sample» можно использовать приведенный ниже способ.

```
Application.Intersect(Range("Test"), Range("Sample")).Select
```

Чтобы этот способ сработал, оба диапазона должны относиться к одному листу.

14. Выбор последней ячейки столбца непрерывных данных

Для выбора последней ячейки в непрерывном столбце можно использовать приведенный ниже способ.

```
ActiveSheet.Range("a1").End(xlDown).Select
```

Если выполнить этот код для приведенной выше таблицы-примера, будет выбрана ячейка A4.

15. Выбор пустой ячейки, расположенной ниже непрерывного столбца

Для выбора ячейки, расположенной ниже непрерывного диапазона, можно использовать приведенный ниже способ.

```
ActiveSheet.Range("a1").End(xlDown).Offset(1,0).Select
```

Если выполнить этот код для приведенной выше таблицы-примера, будет выбрана ячейка A5.

16. Выбор полного непрерывного диапазона ячеек в столбце

Для выбора непрерывного диапазона ячеек в столбце можно использовать приведенные ниже способы.

```
ActiveSheet.Range("a1", ActiveSheet.Range("a1").End(xlDown)).Select
```

-или-

```
ActiveSheet.Range("a1:" & ActiveSheet.Range("a1"). _
  End(xlDown).Address).Select
```

Если выполнить этот код для приведенной выше таблицы-примера, будут выбраны ячейки с A1 по A4.

17. Выбор полного прерывающегося диапазона ячеек в столбце

Для выбора прерывающегося диапазона ячеек в столбце можно использовать приведенные ниже способы.

```
ActiveSheet.Range("a1",ActiveSheet.Range("a65536").End(xlUp)).Select
```

-или-

```
ActiveSheet.Range("a1:" & ActiveSheet.Range("a65536"). _
```

End(xlUp).Address).Select

Если запустить этот код для приведенной выше таблицы-примера, будут выбраны ячейки с A1 по A6.

18. Выбор прямоугольного диапазона ячеек

Для выбора прямоугольного диапазона ячеек вокруг определенной ячейки следует использовать метод **CurrentRegion**. При помощи метода **CurrentRegion** выбирается диапазон, ограниченный пустыми строками и столбцами в любом сочетании. Применение метода **CurrentRegion** поясняется приведенным ниже примером.

```
ActiveSheet.Range("a1").CurrentRegion.Select
```

Выполнение этого кода приводит к выбору ячеек с A1 по C4. Ниже приведены другие способы выбора того же диапазона ячеек.

```
ActiveSheet.Range("a1", _
 ActiveSheet.Range("a1").End(xlDown).End(xlToRight)).Select
```

-или-

```
ActiveSheet.Range("a1:" & _
 ActiveSheet.Range("a1").End(xlDown).End(xlToRight).Address).Select
```

В некоторых случаях может понадобиться выбрать ячейки A1-C6. В данном примере метод **CurrentRegion** не сработает из-за пустой строки 5. Приведенные ниже примеры позволяют выбрать все ячейки.

- lastCol = ActiveSheet.Range("a1").End(xlToRight).Column
- lastRow = ActiveSheet.Cells(65536, lastCol).End(xlUp).Row
- ActiveSheet.Range("a1", ActiveSheet.Cells(lastRow, lastCol)).Select

-или-

```
lastCol = ActiveSheet.Range("a1").End(xlToRight).Column
lastRow = ActiveSheet.Cells(65536, lastCol).End(xlUp).Row
ActiveSheet.Range("a1:" & _
 ActiveSheet.Cells(lastRow, lastCol).Address).Select
```

Примечания к примерам

- Свойство **ActiveSheet** обычно можно опускать, поскольку оно используется по умолчанию, если не указан определенный лист. Например, вместо кода

```
ActiveSheet.Range("D5").Select
```

можно написать

```
Range("D5").Select
```

- Свойство **ActiveWorkbook** также может быть опущено в большинстве случаев. Если не указана конкретная книга, по умолчанию используется активная книга.
- Если при использовании метода **Application.Goto** нужно вызвать два метода **Cells** в методе **Range**, когда указанный диапазон относится к другому (неактивному) рабочему листу, необходимо каждый раз использовать объект **Sheets**, например:

```
Application.Goto Sheets("Sheet1").Range( _
 Sheets("Sheet1").Range(Sheets("Sheet1").Cells(2, 3), _
```

Sheets("Sheet1").Cells(4, 5)))

- Вместо любого элемента в кавычках (например, именованного диапазона «Test») можно использовать переменную со строковым значением. Например, вместо кода

  ActiveWorkbook.Sheets("Sheet1").Activate

  можно написать

  ActiveWorkbook.Sheets(myVar).Activate

  где переменная myVar имеет значение «Sheet1».

Связь объекта Range и свойства Cells

Так как ячейка является частным случаем диапазона, состоящим только из единственной ячейки, объект Range также позволяет работать с ней. Объект Cells (ячейки) – это альтернативный способ работы с ячейкой. Например, ячейка A2 как объект описывается Range («A2») или Cells (l,2). В свою очередь, объект cells, вкладываясь в Range, также позволяет записывать диапазон в альтернативном виде, который иногда удобен для работы, а именно Range(«A2:C3») и Range(Cells(1,2), Cells(3,3)) определяют один и тот же диапазон.

Методы объекта Range, использующие команды Excel

Встроенные в Excel команды и методы позволяют эффективно работать с диапазоном: заполнять его элементами по образцу, сортировать, фильтровать и консолидировать данные, строить итоговую таблицу и создавать сценарии, решать нелинейное уравнение с одной переменной. Полный список методов объекта Range приведен в приложении 1.

Метод AutoFill

Метод AutoFill (автозаполнение) автоматически заполняет ячейки диапазона элементами последовательности. Метод AutoFill отличается от метода DataSeries тем, что явно указывается диапазон, в котором будет располагаться прогрессия. Вручную этот метод эквивалентен расположению указателя мыши на маркере заполнения выделенного диапазона (в который введены значения, порождающие создаваемую последовательность) и протаскиванию маркера заполнения вдоль диапазона, в котором будет располагаться создаваемая последовательность.

Синтаксис:

объект. AutoFill(диапазон, тип)

Аргументы:

Диапазон Диапазон, с которого начинается заполнение тип Допустимые значения: xlFillDefault, xlFillSeries, xlFillCopy, xlFillFormats, xlFillValues,xlFillDays, xlFillWeekdays, xlFillMonths, xlFillYears, xlLinearTrend, xlGrowthTrend. По умолчанию xlFillDefault

Метод AutoFilter

Метод AutoFilter (автофильтр) представляет собой простой способ запроса и фильтрации данных на рабочем листе. Если AutoFilter активизирован, то каждый заголовок поля выделенного диапазона данных превращается в поле с раскрывающимся списком. Выбирая запрос на вывод данных в поле с раскрывающимся списком, осуществляется вывод только тех записей, которые удовлетворяют указанным условиям. Поле с раскрывающимся списком содержит следующие типы условий: Все (All), Первые десять (Top 10), Условие (Custom), конкретный элемент данных, Пустые (Blanks) и Непустые (NonBlanks). Вручную метод запускается посредством выбора команды Данные, Фильтр, Автофильтр (Data, Filter, AutoFilter).

При применении метода AutoFilter допустимы два синтаксиса.

Синтаксис 1:

Объект. AutoFilter

В этом случае метод AutoFilter выбирает или отменяет команду Данные, Фильтр, Автофильтр (Data, Filter, AutoFilter), примененную к диапазону, заданному в аргументе объект.

Синтаксис 2:

Объект. AutoFilter (field, criteria1, operator, criteria2)

В этом случае метод AutoFilter выполняет команду Данные, Фильтр, Автофильтр (Data, Filter, AutoFilter) по критериям, указанным в аргументе.

Аргументы:

field Целое, указывающее поле, в котором производится фильтрация данных

Criteria1 Задают два возможных условия фильтрации и criteria2 поля. Допускается использование строковой постоянной, например 101, и знаков отношений >, <,>=, <=, =, <>

operator Допустимые значения: X1And (логическое объединение первого и второго критериев); X1or (логическое сложение первого и второго критериев)

При работе с фильтрами полезны метод showAllData и свойства FilterMode и AutoFilterMode.

Метод ShowAllData

Показывает все отфильтрованные и неотфильтрованные строки рабочего листа

свойство FilterMode Допустимые значения: True (если на рабочем листе имеются отфильтрованные данные со скрытыми строками), False (в противном случае)

Свойство AutoFilterMode Допустимые значения: True (если на рабочем листе выведены раскрывающиеся списки метода AutoFilter), False (в противном случае)

ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ

В конце XX века широкое распространение получило визуальное программирование – технология, предоставляющая программисту наглядные средства конструирования интерфейса. Объектно-ориентированное программирование удачно использует концепцию визуального программирования.

VBA – это интегрированная среда разработки, которая предоставляет программисту возможность создания форм, на которых размещают компоненты (в терминах VBA - controls, элементы управления), имеющиеся в библиотеке VBA или созданные пользователем. Все компоненты (формы, элементы управления, меню и панели инструментов) являются объектами со своими свойствами и методами и способны реагировать на определенные события.

Компоненты могут быть:

визуальными – т.е. видимыми при работе приложения; немедленно отображаются на экране при проектировании в таком же виде, в каком их увидит пользователь во время выполнения приложения;

не визуальными – отображаются на форме в процессе проектирования в виде значка, но пользователю во время выполнения программы не видны; выполняют некоторые служебные функции.

Использование визуального проектирования интерфейса предоставляет программисту (пользователю) следующие преимущества:

можно легко изменять размеры и расположение компонентов на форме (с помощью простых манипуляций мышью);

в процессе проектирования постоянно виден результат – изображение формы и расположенных на ней компонентов (не надо запускать приложение для проверки внешнего вида окна и последующего изменения программного кода для подбора более удачного размера и расположения компонентов);

(основное) во время проектирования формы и размещения на ней компонентов редактор кода автоматически генерирует код программы, включая в нее фрагменты, описывающие данный компонент (далее можно изменять свойства компонентов и писать обработчики событий).

Визуальное проектирование приложения состоит из следующих этапов:

- создание пользовательской формы;
- размещение на созданной форме нужных компонентов (элементов управления);
- задание определенных свойств этих компонентов;
- написание, при необходимости, обработчиков событий.

Создание пользовательских форм

Объектно-ориентированное программирование позволяет создавать графический интерфейс разрабатываемых приложений на основе использования управляющих элементов. Элементы управления являются объектами. Поэтому, как любые объекты, они обладают свойствами, методами и могут откликаться на события. Элементы управления можно вставлять как в рабочие листы, так и в экранные формы. Экранные формы (их также называют пользовательскими формами, от англ. UserForm) — это окна (обычного вида либо диалоговые), являющиеся частью интерфейса приложения.

Работа с пользовательской формой состоит из нескольких этапов:

- Открыть или создать файл для работы;
- Перейти в редактор Visual Basic;
- Создать пользовательскую форму;
- Применить к созданной форме свойства;

- Создать в форме элементы управления;
- Написать процедуры обработки событий.

Пользовательские формы обычно нужны, когда требуется запросить информацию у пользователя. Разнообразные диалоги настроек, параметров и т.д. Такие окна называются пользовательскими формами.

Чтобы создать форму, откройте редактор VBA (Alt + F11), выберите в списке слева Microsoft Excel Objects и в его меню выбираем Insert->UserForm (рис.2).

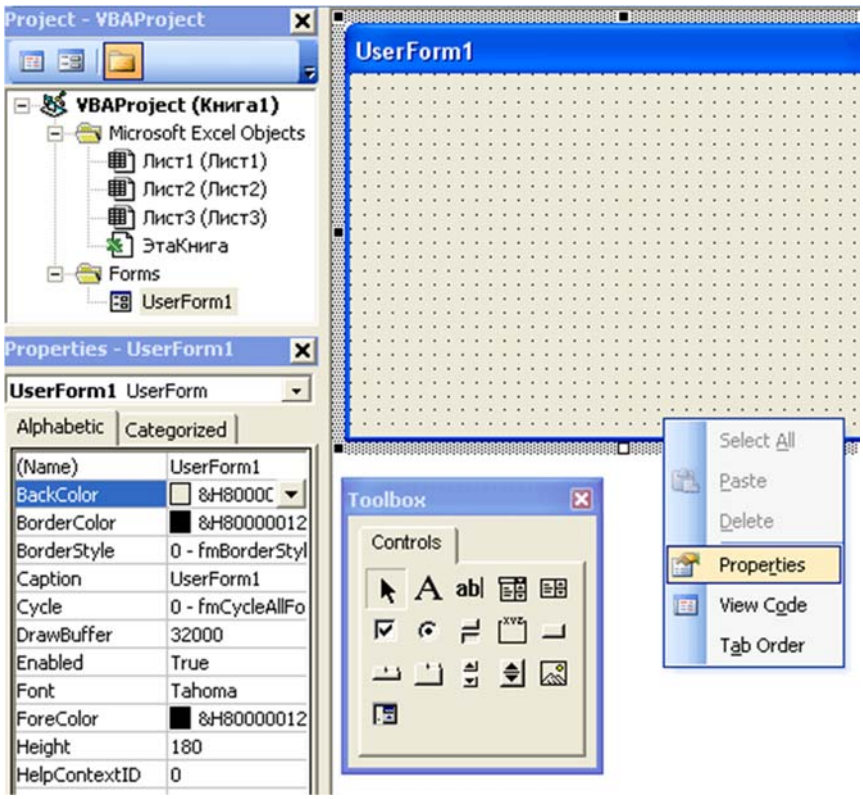

Рис.5. Добавление пользовательской формы.

На экране появилась пустая форма. В ее контекстном меню выбираем Properties - открывается окно свойств формы. В этом окне можно изменить заголовок формы, цвет фона, шрифт и многие другие параметры.

Свойства и методы объекта UserForm

Объект UserForm обладает большим списком свойств. Наиболее часто используемые свойства объекта UserForm представлены в таблице 1. Методы объекта UserForm, которые чаще всего используются для написания пользовательских приложений представлены в таблице 2.

Таблица 1

Свойства объекта UserForm.

Свойство	Описание
ActiveControl	Возвращает объектную ссылку на элемент управления, находящийся в фокусе в данный момент. Свойство только для чтения
BackColor	Возвращает целое значение типа Long, которое определяет цвет фона формы
BorderStyle	Устанавливает тип границы
Caption	Возвращает текст, отображаемый в строке заголовка формы
Controls	Возвращает коллекцию всех элементов управления формы. Только для чтения
Cycle	Определяет, должно ли нажатие клавиши табуляции вызывать последовательный выбор всех элементов управления во всех группах и на каждой странице многостраничных элементов управления или только в пределах текущей группы или страницы. Может принимать значение одной из встроенных констант: fmCycleAllForms или fmCycleCurrentForms
Enabled	Содержит значение типа Boolean, указывающее, доступна ли форма. Если его значение равно False, ни один из элементов управления формы не доступен
Font	Позволяет выбрать параметры шрифта формы или элемента управления
ForeColor	То же самое, что и свойство BackColor, но устанавливает цвет, используемый для переднего плана (обычно это цвет текста) объекта формы
Height и Width	Возвращают высоту и ширину формы в пунктах
Left и Top	Возвращают местоположение левого верхнего угла формы в пунктах
Name	Возвращает имя пользовательской формы
Picture	Указывает рисунок, отображаемый как фон формы
StartUpPosition	Возвращает значение, определяющее положение формы при ее первом отображении на экране. Допустимые значения: • Manual – начальное значение не устанавливается; • CenterOwner – выравнивание по центру объекта, которому принадлежит форма; • CenterScreen – выравнивание по центру экрана; • Windows Default – положение верхнего левого угла экрана

Таблица 2

Наиболее часто используемые методы объекта UserForm

Метод	Назначение
Copy	Копирует выделенный в элементе управления текст в буфер обмена Windows
Cut	Вырезает выделенный в элементе управления текст и помещает его в буфер обмена Windows
Hide	Скрывает форму, не выгружая ее из памяти, сохраняя значения элементов управления формы и всех переменных, объявленных в модуле класса формы
Move	Изменяет положение и размер формы
Paste	Вставляет содержимое буфера обмена Windows в текущий элемент управления
PrintForm	Выводит на используемый в Windows по умолчанию принтер изображение формы, включая все данные, введенные в элементы управления
Repaint	Перерисовывает форму, выведенную на экран. Используйте этот метод, если хотите перерисовать форму, не ожидая, когда она будет перерисована через обычный период времени
Show	Выводит форму на экран. Если форма еще не загружена в память, то данный метод сначала ее загружает

Основная часть кода, который разработчик записывает в модуль класса формы, связана с обработкой событий. В таб.14 перечислены соновные события объекта UserForm, для которых можно написать обработчик в режиме визуального программирования.

Таблица 3

Основные события объекта UserForm

Событие	Описание
Activate	Происходит, когда окно формы становится активным. Используйте это событие для обновления содержимого диалоговых элементов управления, чтобы отразить любые изменения, которые произошли, пока окно формы было неактивным
Click	Происходит при щелчке мышью по форме (любой ее части, не занятой элементами управления)
DblClick	Происходит при двойном щелчке мышью по форме (любой ее части, не занятой элементами управления)
Deactivate	Происходит, когда форма перестает быть активной
Initialize	Происходит, когда форма впервые загружается в память посредством выполнения оператора Load или с помощью метода Show. Используйте это событие для инициализации элементов управления формы при ее появлении на экране
Resize	Происходит при изменении размеров формы
Terminate	Происходит при закрытии формы, т.е. когда форма выгружается из памяти. Используйте это событие для осуществления специальных задач, которые необходимо выполнить прежде, чем переменные формы будут выгружены

В дополнение к методам, свойствам и событиям, встроенным в объект UserForm, VBA предоставляет два оператора, которые особенно полезны при работе с объектами форм: Load и Unload.

Синтаксис операторов Load и Unload:

Load Object Unload Object

Здесь Object представляет любую допустимую ссылку на объект UserForm.

Оператор Load загружает в память объект UserForm и запускает метод формы Initialize, но не выводит форму на экран. Когда форма загружена, можно использовать написанную на VBA программу для работы с объектом UserForm.

Оператор Unload удаляет из памяти объект UserForm, а также все переменные формы. После того, как форма выгружена, она перестает быть доступной для VBA-кода.

Кроме окна свойств есть еще инструментальная панель Toolbox. В ней содержатся элементы управления, которые можно разместить на форме. При создании экранной формы автоматически отображается “Панель инструментов”. “Панель инструментов” содержит элементы управления, которые можно использовать в экранной форме. Если панель инструментов не появилась при создании экранной формы, то вызвать элементы управления можно, используя команду View | ToolBox или нажав на кнопку  .

“Панель инструментов” (или “Панель элементов управления”) предназначена для создания и редактирования объектов вашего приложения. При написании программ на VBA вы обязательно столкнетесь с английскими названиями элементов управления, поэтому в таблице даны русские и английские названия элементов управления.

Панель инструментов и элементы управления формы.

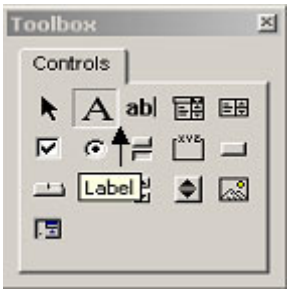

Рис.6. Панель элементов управления ToolBox

Label (надпись) — статическая область текста, обычно содержащая какую-либо поясняющую информацию, а также это поле часто используют для вывода полученных результатов.

TextBox (поле) — элемент для ввода текста пользователем, который в последующем используется в программе.

CommandButton (кнопка управления) — элемент, с помощью которого в пользовательскую форму можно вставить командную кнопку. При нажатии на командную кнопку выполняются запрограммированные вами действия.

ListBox (список) — применяется для хранения списка значений. В списке пользователь может выбрать одно или несколько значений, которые в дальнейшем используются в программе.

ComboBox (поле со списком) — применяется для хранения списка значений. Этот элемент сочетает возможности элементов ListBox и TextBox.

ScrolBar (полоса прокрутки) как элемент управления — это не совсем то, чем вы привычно пользуетесь для прокрутки, например, рабочего листа. В данном случае полосы

прокрутки применяются в качестве механизмов выбора. В графических программах вы, наверное, видели, как с помощью полос прокрутки можно выбрать цвет либо сделать изображение более светлым или более темным.

SpinButton (счетчик) — используется для ввода или изменения числовых значений.

OptionButton (переключатель) — позволяет выбрать один из нескольких взаимоисключающих параметров или действий. Переключатели обычно отображаются группами, обеспечивая возможность альтернативного выбора.

CheckBox (флажок) — предоставляет пользователю возможность выбора. Флажок обычно имеет два состояния: установленное и сброшенное.

Togglebutton (выключатель) — кнопка, которая остается нажатой после щелчка на ней, и возвращается в исходное состояние после повторного щелчка.

Frame (рамка) — используется для визуального объединения каких-либо элементов управления в группу, показывая, что эти элементы связаны между собой.

Image (рисунок) — позволяет вставлять графические элементы в экранные формы. С помощью этого элемента можно вставлять изображения из графических файлов следующих типов: bmp; cur; gif; ico; jpg; wmf.

MultiPage (набор страниц) — этот элемент управления внешне похож на набор вкладок и также содержит одну или несколько страниц. Отличие между ними заключается в том, что страницы являются формами, содержащими собственные элементы управления (включая наборы вкладок), которые можно отформатировать всеми средствами форматирования экранных форм.

TabStrip (набор вкладок) — элемент управления, который содержит одну или несколько вкладок. Используется для организации в группы связанной информации.

RefEdit (поле со свертыванием) — этот элемент похож на обычное поле ввода, но имеет кнопку с правой стороны поля, которая сворачивает экранную форму, что позволяет легко выбрать любой диапазон ячеек на рабочем листе.

В таблице 4 представлены основные элементы управления формы.

Таблица 4

Стандартные элементы управления, включенные в VBA

Элемент управления	Назначение	Рисунок
Label (надпись, метка)	Позволяет создавать заголовки элементов управления, которые не имеют собственных встроенных заголовков	
TextBox (текстовое поле)	Окно редактируемого текста свободной формы для ввода данных. Может быть одно- и многострочным	
ComboBox (поле со списком)	Объединяет окно редактирования и окно списка	
ListBox (список)	Отображает список значений, из которых пользователь может сделать выбор	
CheckBox (флажок)	Стандартный флажок, который используется для выбора вариантов, не являющихся взаимоисключающими	
OptionButton (переключатель)	Стандартная кнопка-переключатель. Используется, когда пользователю необходимо сделать выбор между "включено/выключено"	
ToggleButton (выключатель)	Выключатели служат для той же цели, что и флажки, но выводят установки в виде кнопки, находящейся в "нажатом" или "отжатом" состоянии	
Frame (рамка)	Визуально и логически объединяет некоторые элементы управления (особенно флажки, переключатели и выключатели)	
CommandButton (кнопка)	Используется для выполнения таких действий, как Cancel (Отмена), Save (Сохранить), Ok и т.д. Когда пользователь щелкает по кнопке, выполняется VBA-процедура, закрепленная за данным элементом управления	
TabStrip (набор вкладок)	Состоит из области, в которую следует помещать другие элементы управления (такие, как текстовые поля, флажки и т.д.)	
MultiPage (набор страниц)	Состоит из нескольких страниц. Можно выбрать любую из них, щелкнув по соответствующей вкладке	
ScrollBar (полоса прокрутки)	Позволяет выбирать линейное значение, аналогичное тому, как это можно сделать при помощи счетчика	
SpinButton (счетчик)	Специальная разновидность текстового поля. Используется для ввода последовательных величин, которые заведомо находятся в определенном интервале значений (число, дата и т.п.)	
Image (рисунок)	Выводит на форме графическое изображение любом из следующих форматов: *.bmp, *.cur, *.gif, *.ico, *.jpg, *.wmf	

Для удобства работы с элементами управления в период их конструирования в приложениях Microsoft Office введен режим конструктора, который активизируется нажатием кнопки Режим конструктора (Designe Mode) панели инструментов. В

режиме конструктора отключена реакция элемента управления на события. Поэтому при включенном режиме конструктора можно видоизменять элемент управления и задавать его свойства. Размещенный на форме элемент управления можно перемещать, изменять его размеры, копировать в буфер обмена и вставлять из буфера обмена. Отключается режим конструктора той же кнопкой Режим конструктора (Designe Mode).

Работа программы основывается на реакции объекта на какое-либо событие. Каждому событию объекта можно назначить процедуру обработки данного события. В приложении 3 представлены основные события элементов управления.

Для нашей работы мы будем использовать не все элементы управления. Наиболее часто используемыми являются:

- Label (надпись);
- TextBox (поле);
- CommandButton (кнопка управления);
- Image (рисунок);
- OptionButton (переключатель);
- CheckBox (флажок).

Например, можно добавить элемент управления Кнопка – CommandButton:

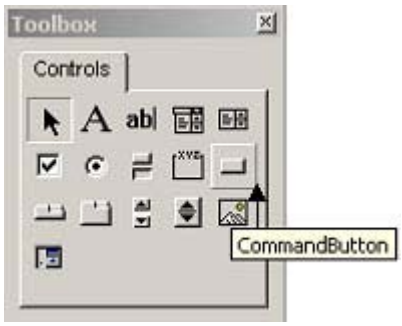

Рис. 7. Элемент управления CommandButton - кнопка

Теперь в окне свойств доступны свойства кнопки. Можно изменить надпись на ней, или добавить картинку. Осталось только привязать к этой кнопке свой код.

Это очень просто - по двойному щелчку на кнопке вы попадете в окно редактора VBA, где уже создана процедура обработки нажатия на кнопку.

```
Private Sub CommandButton1_Click()
End Sub
```

Важно: По двойному щелчку на любом элементе управления создается процедура обработки этого объекта по умолчанию. Список остальных событий, на которые может реагировать элемент управления можно увидеть из раскрывающегося списка в правом верхнем углу (рис 4)

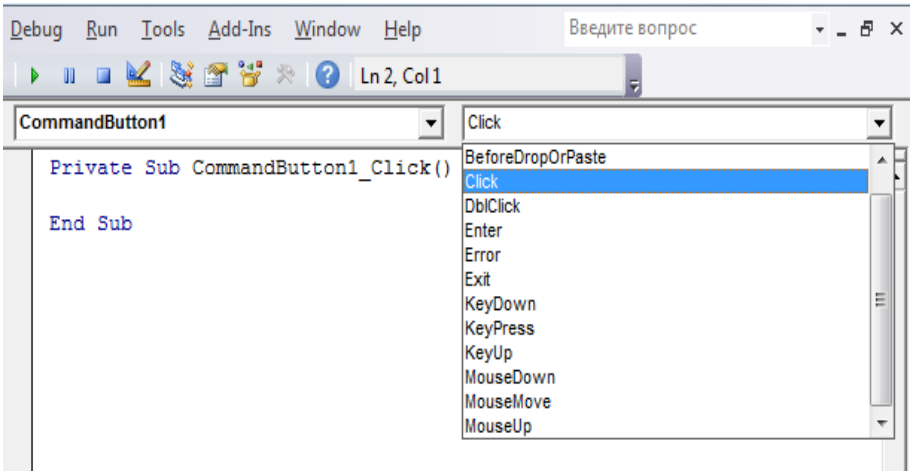

Рис. 8. Список событий элемента управления

Добавление в список нового элемента

Для добавления в список нового элемента следует использовать метод **AddItem**. В случае если список состоит из нескольких строк, то к нему будет добавлена новая строка.

Синтаксис: **Object.AddItem (Item [.Index])**

Object - объектная переменная одного из следующих типов: **Combobox** или **ListBox.**

Item - определяет элемент или строку списка, которая будет добавлена. Номер первого элемента или строки соответствует 0, второго – 1 и так далее.

Index - определяет позицию добавляемого элемента в списке. Значением данного свойства может быть целое число, которое не должно превышать число элементов управления.

Пример:

Private Sub *UserForm1_Activate ()*

*ListBox1.**AddItem** (“Item1”)*
*ListBox1.**AddItem** (“Item2”)*
*ListBox1.**AddItem** (“Item3”)*

End Sub

ОБЪЯВЛЕНИЕ ПОЛЬЗОВАТЕЛЬСКИХ КЛАССОВ В ПРИМЕРАХ

Класс - это шаблон, на основе которого во время выполнения программы создается объект.

Концептуально, создание класса начинается с этапа проектирования, где определяются свойства и методы, которыми должны обладать объекты нового класса.

Рассмотрим следующий пример: пусть требуется создать класс, описывающий студента. Объект такого класса может представлять в виде свойств такую информацию о студенте, как его имя, отчество и фамилию, дату и место рождения, контактную информацию, пол, паспортные данные и т.п. В качестве методов можно указать возможность редактирования информации, возможность полного или, наоборот, частичного отображения информации о студенте. Семантически такое описание может быть представлено, например, следующим образом:

```
// псевдокод описания объекта Студент
Студент {
 Фамилия: строка
 Имя: строка
Отчество: строка
Дата_рождения: Дата
 Пол: символьный или логический
 Контактная_информация: строка
}
Студент.Полное_имя: строка, только чтение
Студент.Полное_описание: строка, только чтение
Студент.Править_описание (Нов_Фамилия, Нов_Имя, Нов_Отчество, Нов_Дата_рожд, _
Нов_Пол, Нов_Конт_Инф)
```

Как видно, такое описание очень напоминает описание пользовательского типа данных. Однако отличие объектного типа в том, что не только данные (Фамилия, Имя и пр.), но и код (Студент.Полное_имя и пр.) размещены совместно (инкапсулированы). Синтаксически, объявление класса в VBA выполняется в специализированной программной единице - модуле класса, куда помещается весь код создаваемого класса. Отметим, что в VBA не существует специальных языковых конструкций, используемых при описании класса (кроме описания свойств), поэтому порядок создания класса удобнее рассмотреть по шагам, а затем прокомментировать:

В редакторе Visual Basic добавляем в проект новый модуль класса (меню "Вставка/Модуль класса"). Откроется окно нового модуля класса.

В окне свойств (F4) задаем имя (Name) модуля (в нашем примере - CStudent). Оно же станет именем класса.

В разделе описаний модуля объявляем закрытые члены класса - обычные переменные уровня модуля (Private), которые будут определять значения свойств.

Инициализируем начальные значения свойств при помощи метода Class_Initialize (конструктора класса).

Определяем свойства для чтения (Property Let), для записи (Property Let) и объектные свойства (Property Set).

Создаем методы класса при помощи обычных процедур и функций.

Допустимо также создание метода Class_Terminate (деструктора класса) для удаления объекта из памяти по завершении работы с ним.

По этому алгоритму был создан ранее спроектированный класс CStudent, представляющий описание объекта "студент". Полный код модуля с комментариями приведен в листинге 24.

Листинг 24. Модуль класса

```
' Закрытые члены класса, прямой доступ к ним невозможен – только через свойства

Private fLastName As String ' фамилия
Private fFirstName As String ' имя
Private fMiddleName As String ' отчество
Private fBirthDay As Date ' дата рождения
Private fGender As String * 3 ' пол
Private fContacts As String ' адрес и телефон
' Конструктор класса. Вызывается в момент создания объектной переменной
Private Sub Class_Initialize()
 fLastName = "Фамилия не указана"
 fFirstName = "Имя не указано"
 fMiddleName = "Отчество не указано"
 fContacts = "Адрес не указан"
End Sub
' Свойства для чтения
Public Property Get LastName() As String
 LastName = fLastName
End Property
Public Property Get FirstName() As String
 FirstName = fFirstName
End Property
Public Property Get MiddleName() As String
 MiddleName = fMiddleName
End Property
Public Property Get BirthDay() As Date
 BirthDay = fBirthDay
End Property
Public Property Get Gender() As String
 Gender = fGender
End Property
Public Property Get Contacts() As String
 Contacts = fContacts
End Property
' Свойства для записи
Public Property Let LastName(ByVal NewValue As String)
 fLastName = NewValue
End Property
```

```
Public Property Let FirstName(ByVal NewValue As String)
 fFirstName = NewValue
End Property
Public Property Let MiddleName(ByVal NewValue As String)
 fMiddleName = NewValue
End Property
Public Property Let BirthDay(ByVal NewValue As Date)
 fBirthDay = NewValue
End Property
Public Property Let Gender(ByVal NewValue As String)
 fGender = NewValue
End Property
Public Property Let Contacts(ByVal NewValue As String)
 fContacts = NewValue
End Property
' Свойства ТОЛЬКО ДЛЯ ЧТЕНИЯ (нет парных Property Let)
Public Property Get FullName() As String
 FullName = fLastName & " " & fFirstName & " " & fMiddleName
End Property
Public Property Get FullInfo() As String
 FullInfo = fLastName & Chr(9) & fFirstName & Chr(9) & fMiddleName & Chr(9) _
 & CStr(fBirthDay) & Chr(9) & fGender & Chr(9) & fContacts
End Property
' Метод для изменения свойств объекта.
' Все параметры объявлены как необязательные (Optional).
 Изменять поля можно и через соответствующие свойства для записи (Property Let),
' этот метод приведен для примера
Public Sub EditInfo(Optional LastName As String, Optional FirsName As String, _
 Optional MiddleName As String, Optional BirthDay As Date, _
 Optional Gender As String, Optional Contacts As String)
 fLastName = LastName
 fFirstName = FirstName
 fMiddleName = MiddleName
 fBirthDay = BirthDay
 fGender = Gender
 fContacts = Contacts
End Sub
```

Дополнительные пояснения к коду листинга 24 в части синтаксиса:

Синтаксис описания членов класса такой же, как и для обычных переменных. Использование Private позволяет ограничить доступ к членам класса из других модулей и, тем самым, обеспечить целостность данных.

Конструктор Class_Initialize предназначен для задания начальных значений переменным - членам класса и выполнения других операций инициализации. Он вызывается автоматически - в момент создания экземпляра класса.

Свойства класса объявляются с ключевым словом Property. При этом функция Property Get создает свойство для чтения, а процедура Property Let служит для записи значений в свойства базовых типов (в т.ч. массивов и пользовательских типов). Процедура Property Set служит для присваивания значений членам объектного типа (см. листинг 25). Общий синтаксис объявления свойств:

```
' для чтения
[Public | Private] [Static] Property Get <имяФункции>
[(<списокАргументов>)] [As <тип>]
 <операторы>
имяФункции = возвращаемое значение
 [Exit Property]
 <операторы>
имяФункции = возвращаемое значение
End Property
' для записи
[Public | Private] [Static] Property Let <имяПроцедуры> [(<списокАргументов>)]
 <операторы>
 [Exit Property]
 <операторы>
End Property
' для работы с объектными членами класса
[Public | Private] [Static] Property Set <имяПроцедуры>
[(<списокАргументов>)]
<операторы>
 [Exit Property]
 <операторы>
End Property
```

Методы класса - это обычные процедуры и функции VBA, которые объявляются в модуле класса и не могут быть использованы самостоятельно.

VBA поддерживает один из основных принципов ООП, наследование, косвенным путем - через встраивание объектов. Т.е. нет прямой возможности создавать классы-потомки на основе ранее созданных классов, но можно объявлять членами класса переменные объектных типов. Пример встраивания приведен в листинге 31, где на основе классов CStudent (Студент) и CFaculty (Факультет) создается класс CGroup, описывающий учебную группу указанного факультета.

Листинг 25. Встраивание

```
' ----------------------------------------------
' Модуль класса CFaculty - объект "Факультет" (приведен фрагмент кода)
' ----------------------------------------------
Private fTitle As String ' Название факультета
Public Property Get Title() As String
 Title = fTitle
End Property
Public Property Let Title(NewTitle As String)
 fTitle = NewTitle
```

```
        End Property
        ...
'                                                    ------------------------------------------
' Модуль класса CGroup - объект "Учебная группа" (приведен фрагмент кода)
' ------------------------------------------
        ' Встраивание объектных членов класса
        Private fFaculty As New CFaculty
        Dim fStudents() As New CStudent
        ...
' Использование Property Set для задания значения объектному члену
        Public Property Set Faculty(Title)
         Set fFaculty = Title
        End Property
        ' Чтение названия факультета
        Public Property Get Faculty() As String
         Faculty = fFaculty.Title
        End Property
        ...
```

Создание объектных переменных

Поскольку класс - не более чем специфичный тип данных, то для его использования в программе требуются переменные, представляющие экземпляры этого класса. Такие переменные, называемые объектными, создаются одним из способов:

явным указанием класса объекта;

ссылкой на ранее созданный объект.

При любом способе создания объектная переменная представляет из себя 4-байтовую ссылку на адрес, где хранится объект. При объявлении такой переменной память для самого объекта может и не отводиться, поэтому может быть не определено и значение ссылки. Задание ссылки на объект, т.е. связывание объектной переменной с самим объектом выполняется двумя способами:

При раннем связывании в момент объявления указывается класс объекта:

Dim <Переменная> As <классОбъекта>

Это позволяет еще на этапе трансляции проверять, допустимы ли те или иные операции над создаваемыми объектами.

При позднем связывании переменная объявляется так:

Dim <Переменная> As Object

Это объявление говорит о том, что переменная является объектом (ссылкой), но ничего не сказано о классе этого объекта. Он выяснится только динамически при выполнении программы, когда <Переменная> будет связываться с только что созданным или существующим объектом того или иного класса. Поэтому такое связывание и называется поздним, или динамическим.

В целом же, объявление объектных переменных отличается от обычных только указанием ключевого слова New, с помощью которого создаваемому экземпляру класса (при раннем связывании) выделяется память и вызывается его конструктор. Объектные переменные могут быть объявлены и использованы в любых модулях (как в стандартных, так и модулях класса). Общий синтаксис объявления объектной переменной:

Private | Public | Dim <имяОбъектнойПеременной> As New <имяКласса>

Например:

```
Public Faculty As New CFaculty
Private Groups(3) As New CGroup
Dim stud As New CStudent
```

Использование раннего связывания имеет одно преимущество: явное указание класса позволяет получить доступ к его свойствам и методам уже на этапе разработки в VBE. Это выражается в том, что при введении имени объектной переменной появляется всплывающий список доступных операций над объектом.

Использование объектов

Для обращения к свойствам или методам экземпляра класса в VBA используется точечная нотация:

```
перем = <имяОбъектнойПеременной>.<Свойство> ' чтение свойства
```

Или:

```
<имяОбъектнойПеременной>.<Свойство> = значение ' запись свойства
```

Для обращения к объектным свойствам следует использовать ключевое слово Set. Вызов методов и передача параметров аналогичны работе с обычными процедурами и функциями. Примеры работы с экземплярами созданных классов - в листинге 26.

Листинг 26. Работа с объектами

```
' ----------------------------------------------------------------
' Создание экземпляра класса и работа с его свойствами и методами
' ----------------------------------------------------------------
Sub sample30()
 Dim stud As New CStudent ' экземпляр класса CStudent
 ' Обращения к свойствам объекта
 stud.FirstName = "Иван"
 stud.LastName = "Петров"
 stud.Contacts = "г.Омск, пр.Мира, 11, к.8. т/ф (3812) 65-96-11"
End Sub
```

возвращенный свойством CurrentTimeZone.

Наследование в VBA и отношения между объектами

Связь объектов друг с другом может быть различной. Основными видами связи являются hierarchical и containment.

Иерархическая связь

При иерархической связи(hierarchical relationship) одни классы являются наследниками других, более фундаментальных классов. Иерархии классов удобны при описании объектов — подтипов более универсальных классов.

Производные классы наследуют члены класса, от которого они произведены, позволяя добавлять функциональность по мере продвижения вглубь иерархии классов.

Компиляция кода

Убедитесь, что компилятор может получить доступ к классу, который используется для получения нового класса. Это может означать полное уточнение его имени, как в предыдущем примере, или определение его пространства имен в операторе Imports (пространство имен .NET и тип). Если класс находится в другом проекте, может потребоваться добавить ссылку на этот проект.

Отношение вложенности

Другим способом описания отношений между объектами является containment relationship. Объекты-контейнеры инкапсулируют другие объекты. Например, объект OperatingSystem логически содержит объект Version, который возвращается с помощью его свойства Version. Объект контейнера физически не содержит любые другие объекты.

Коллекции

Один из типов отношения вложенности представлен collections. Коллекции представляют собой упорядоченные группы однотипных объектов. Visual Basic поддерживает определенный синтаксис. Оператор

For Each... Next (Visual Basic), позволяющем перебирать элементы коллекции. Кроме того, коллекции часто позволяют использовать свойство Item для извлечения элементов по индексу или привязки их к уникальной строке. Коллекции более просты в применении, чем массивы, т.к. они позволяют добавлять или удалять элементы без использования индексов. Коллекции просты в обращении и потому часто используются для хранения форм и элементов управления.

ТЕСТИРОВАНИЕ И ОТЛАДКА ПРОГРАММ

Процесс исправления ошибок называется отладкой.

Отладка программ и обработка ошибок всегда выступает как часть процесса разработки. Если бы все программировали идеально, то необходимость отладки программ и обработки ошибок отпала бы сама собой: каждая создаваемая программа выполняла бы все требуемые действия с первого раза. Как правило, такого не бывает, поэтому в большинстве систем разработки имеются инструменты, с помощью которых можно решить проблемы, возникающие в процессе программирования. В VBA также есть средства, которые позволяют либо исключить ошибки при разработке, либо задать отклик на ошибки при выполнении программ.

Отладка программ и обработка ошибок — это не одно и то же, но они тесно связаны друг с другом.

Отладка программ — это проверка и внесение исправлений в программу при ее разработке. Отладка позволяет идентифицировать ошибки, допущенные при программировании. Например, синтаксические ошибки в тексте программы, именах функций и переменных или логические ошибки.

Обработка ошибок — это задание реакции на ошибки, которые возникают во время выполнения программы. Причиной ошибок могут быть как ошибки в самой программе, так и другие обстоятельства, находящиеся вне сферы влияния программиста. Например, отсутствие файлов, к которым происходит программное обращение, отказ аппаратных средств или неправильные действия пользователя.

Невозможно предотвратить возникновение всех ошибок, но следует стремиться к уменьшению их числа. В маленькой программе довольно просто выявить ошибку. Однако по мере увеличения размеров и сложности программ находить их становится все труднее. В таких случаях необходимо пользоваться средствами отладки VBA.

Среда разработки программ на VBA предоставляет пользователю современные удобные средства отладки программы.

Предположим, что уже написан код вашей процедуры. Следующий этап в создании любой процедуры — тестирование написанного кода.

Тестирование — это процесс выполнения процедуры и исследование всех аспектов ее работы.

Цель тестирования — проверить правильность результатов выполнения процедуры и ее реакцию на разнообразные действия пользователя.

Если во время работы процедуры получены неверные результаты вычислений, непредвиденная реакция на те или иные действия пользователя, либо вообще произошла остановка выполнения, то это говорит о том, что в тексте программы имеются ошибки.

Все возможные ошибки можно разделить на три вида:

1. Ошибки компиляции. Возникают, если VBA не может интерпретировать введенный текст, например, при использовании неправильного синтаксиса инструкции или задании неверного имени метода или свойства. Некоторые ошибки компиляции обнаруживаются при вводе инструкции, а другие — только перед выполнением программы. Данный тип ошибок обычно просто идентифицировать и исправить, поскольку VBA выявляет их автоматически, а сами ошибки очевидны.

2. Ошибки выполнения. Возникают при выполнении программы, т.е. после успешной компиляции (рис. 9). Причиной таких ошибок может быть отсутствие данных или неправильная информация (например, данные, введенные пользователем).

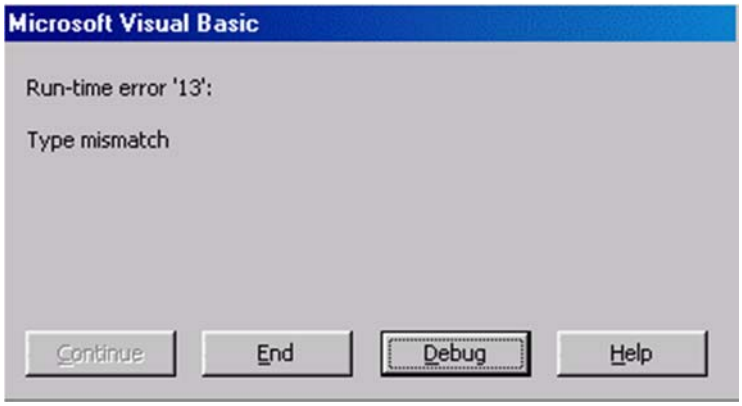

Рис. 9. Сообщение об ошибке времени выполнения программы. Несоответствие присвоенного значения и типа объявленной переменной

Ошибки выполнения, как и ошибки компиляции, легко идентифицируются VBA. При этом выводится инструкция, при выполнении которой произошла ошибка (рис. 10). Ошибки данного типа тяжелее устранить: может понадобиться вывести значения переменных или свойств, а также другие данные, которые влияют на успешное выполнение программы.

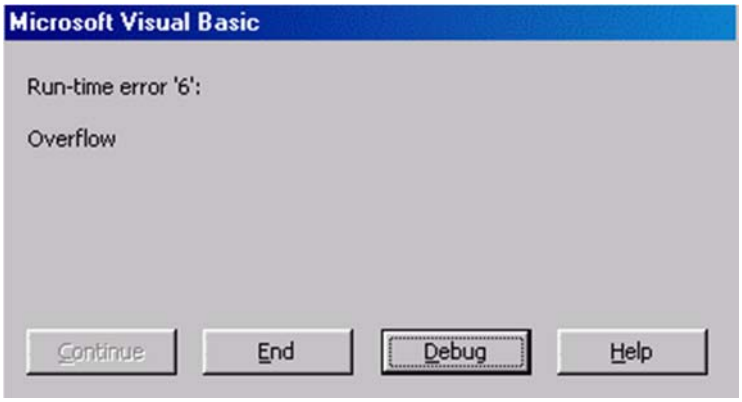

Рис. 10. Сообщение об ошибке выхода за пределы диапазона допустимых значений переменной в ходе выполнения программы

3. Логические ошибки труднее всего заметить и устранить. Логические ошибки не приводят к прекращению компиляции или выполнения. Однако они являются причиной того, что программа не выдает желаемых результатов. Ошибки данного типа идентифицируются путем тщательной проверки с помощью средств отладки VBA.

Компиляция — это процесс преобразования программы, написанной на алгоритмическом языке, в язык машинных кодов. Если в программе есть синтаксические ошибки, то процесс компиляции прекращается, строки с ошибкой закрашиваются желтым цветом и выдается соответствующее сообщение. Для продолжения выполнения программы необходимо исправить ошибку и нажать кнопку “Continue”  на стандартной панели редактора VBA. Или прервать выполнение программы, нажав на кнопку “Reset”  , исправить ошибку в программе, а затем заново запустить ее.

Использование пошаговой трассировки:

Пошаговая трассировка - это метод отладки приложения при котором можно выполнять код по одной команде и следить за ходом её выполнения. Это очень полезный метод! Таким методом можно находить те ошибки, которые не может найти Visual Basic. Такие ошибки называются логическими.

Проверка правильности ввода данных

Ключ к успешному тестированию лежит в словосочетании "здравый смысл", и в следующей истории будет показана жизненная важность этой концепции. Одна

коммунальная компания использовала достаточно сложную, но, как считалось, надёжную программу для посылки счетов на оплату и приема квитанций. В случае когда компания не получала оплаченной квитанции от пользователя, тот автоматически лишался предоставляемых коммунальных услуг. В тот день, когда произошла эта история, один из клиентов компании, уходя в отпуск, выключил электричество. В результате чего компьютер послал ему счёт за электричество в размере $0.00, который, естественно, не мог быть оплачен. После определённого числа запросов на оплату компьютер решил, что тот клиент, которому был послан счет, не уплатил вовремя эти самые $0.00, в следствии чего он остался без электричества!"

В истории не говорится о реакции разгневанного клиента, однако, если эта история произошла на самом деле, то вина программиста, написавшего приложение, в том, что он забыл вставить проверку на наличие пустых счетов. Видимо он просто не предполагал, что подобная ситуация может возникнуть. (Хотя, по правде говоря, многие программисты используют в операторах сравнения выражение a > b, вместо a >= b)."

Для проверки корректности работы программы надо использовать граничные значения, так как именно такие значения позволяют находить ошибки.

При обнаружении ошибки компилятор выдает сообщение с указанием номера ошибки. В этом случае полезно, воспользовавшись справочной системой редактора VBA, определить характер ошибки и исправить ее.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Назовите и проясните основные понятия объектно-ориентированного программирования.
2. В чем заключается преимущество технологии ООП?
3. Что такое программирование, управляемое событиями? В чем его особенности?
4. Каково назначение VBA?
5. Что такое класс? Что объединяет в себе класс?
6. Что такое объект? Приведите примеры объектов.
7. Как можно создать собственный класс в Visual Basic?
8. Из каких элементов состоит модуль класса?
9. Что такое событие?
10. Что такое метод объекта?
11. Какие конструкции применяются для установки свойств объектов и доступа к их методам?
12. Каковы основные характеристики объекта?
13. Что такое конструктор, деструктор?
14. Виды конструкторов класса.
15. В чем отличие порядка выполнения программ в ООП и в процедурном программировании?
16. Каковы основные понятия ООП?
17. Что такое инкапсуляция? Приведите примеры.
18. Что означает наследование? Приведите пример наследования.
19. В чем заключается полиморфизм объектов? Что такое виртуальная функция?
20. Что такое элементы управления? Приведите примеры.
21. Как можно установить на форме элемент управления?
22. Как обеспечить выполнение элементом управления требуемых действий?
23. Каково назначение элементов управления надпись и поле?
24. Какова технология программирования с использованием элементов управления?
25. Что называется коллекцией? Как можно описать коллекцию?
26. Опишите этап формализации задачи на примере.
27. В чем смысл этапа моделирования?
28. Как выполняется разработка интерфейса приложения?
29. Как записать процедуру обработки события?

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Написать на VBA программу под названием «Игра», в которой требуется угадать число, которое случайным образом выдаст программа. В пользовательской форме и процедурах к ней организовать ввод задуманного числа в пределах от 1 до 10-ти и его сравнение с числом, которое случайным образом, в тех же пределах, выдаёт программа. В случае если числа не равны, должно быть выдано сообщение «ВЫ ПРОИГРАЛИ» и значения чисел, а в случае если числа равны - сообщение «ВЫ ВЫИГРАЛИ!!!» и так же значения чисел.

Предусмотреть организацию кнопок для повторения или завершения работы программы и запуска программы с листа Excel.

2. Создать форму для заполнения заявки на туристическую поездку. Обеспечить проверку правильности ввода типов данных, диапазон допустимых значений. Предусмотреть ***зависимые*** поля со списками. Список курортов зависит от выбранной страны, цена за номер зависит от типа номера и т.д. Стоимость проживания рассчитать в зависимости от выбранного типа номера и количества дней проживания. Количество дней рассчитать по датам заезда и выезда. (Функция **DateDiff** используется для вычисления разности двух дат. Возвращает значение типа Variant(Long), указывающее число временных интервалов между двумя датами. Предусмотреть сохранение данных в таблицу Excel , вычисляя последнюю строку текущего заполненного диапазона. Для выполнения задания можно воспользоваться примером (приложение 4).

3. Игра «Поймай хомяка». Создать форму для игрового поля, и передвигающийся случайным образом по этому полю рисованный объект «хомяк». Передвижения объекта замедлить с помощью метода **Application.Wait.** Цель игрока поймать курсором мыши случайно двигающийся объект. При возникновении события щелчка мышью на этом объекте игрок выигрывает. Игру можно усложнить, добавив области допустимой и запрещенной ловли хомяка.

4. Создать форму для игры: в 3-х рядом расположенных контейнерах одновременно с интервалом в несколько миллисекунд меняются подгружаемые картинки из текущей папки приложения, на форме расположен объект кнопка. В случае одновременного появления в трех контейнерах одной и той же картинки, игрок доложен успеть нажать мышью кнопку. При щелчке мышью в момент одновременного появления картинок игрок выигрывает.

ЛИТЕРАТУРА

1. Васильев, А. VBA в Office 2000: учебный курс / А. Васильев, А. Андреев. – СПб.: Изд-во «Питер», 2002. – 432 с.
2. Новиков, Ф.А. Microsoft® Office XP в целом / Ф.А. Новиков, А.Д. Яценко. – СПб.: БХВ-Петербург, 2002. – 928 с.
3. Бадд Т. Объектно-ориентированное программирование в действии / Т. Бадт. - М.: Бином, 1997. – 346 с.
4. Дукин А.Н., Пожидаев А.А.. Самоучитель Visual Basic 2010/ А.Н.Дукин, А.А.Пожидаев.-Петербург.: Изд-во: БХВ-, 2010 - ISBN: 978-5-9775-0512-3
5. Слепцова Л.Д. Программирование на VBA в Microsoft Office 2010 / Л.Д. Слепцова.- Изд-во: Диалектика, 2010 ISBN: 978-5-8459-1663-1
6. Джон Уокенбах. Excel 2013: профессиональное программирование на VBA/ Уокенбах Дж. – Киев: Изд-во Диалектика, 2014. 960 с
7. Питер Эйткен. Разработка приложений на VBA в среде Microsoft Office XP/ Эйткен П ; Киев: Изд-во Вильямс, 2003.- 496 с.
8. Пол Киммел, Джон Грин, Стивен Буллен Microsoft Office Excel 2003 и VBA. Справочник программиста/ Киммел П., Грин Дж, Буллен С . – Киев: Изд-во Диалектика, 2005. 1088 с

ПРИЛОЖЕНИЯ

Приложение 1

Список Объектно-ориентированных языков

- **C#**
- **C++**
- **F#**
- **Java**
- **Delphi**
- **Eiffel**
- **Simula**
- **D**
- **Io**
- **Objective-C**
- **Swift**
- **Object Pascal**
- **VB.NET**
- **Visual DataFlex**
- **Perl**
- **PowerBuilder**
- **Python**
- **Scala**
- **ActionScript (3.0)**
- **JavaScript**
- **JScript .NET**
- **Ruby**
- **Smalltalk**
- **Ada**
- **Xbase++**
- **X++**
- **Vala**
- **PHP**
- **Cyclone**

Примечание: кроме ОО — языков общего назначения существуют и узкоспециализированные ОО — языки.

Приложение 2

Свойства и методы объекта Range

Свойство	Описание
Address, AddressLocals	Возвращает строку, задающую ссылку на Range объект. Во втором случае это ссылка в языке пользователя. Эту ссылку можно выдавать в формате A1 или R1C1, как абсолютную или относительную. Вид возвращаемого значения определяют параметры этого свойства (метода).
Areas	Применимо обычно к объекту Selection и возвращает коллекцию объектов Range в случае, когда Selection (Range) задает несвязную область. Возвращается сам объектRange, если область содержит только один объект.
Borders	Возвращает коллекцию из четырех границ объекта Range. Позволяет выделить цветом и (или) толщиной линии границы объекта.
Text, Characters	Свойство Text возвращает строку текста, связанного с Range объектом (ячейкой). Имеет статус только для чтения. Если нужно изменить весь текст или его часть, то можно использовать свойство (метод) Characters, два параметра которого: Start иLength позволяют выделить требуемую подстроку текста.
Column, Row	Возвращают соответственно номер первого столбца или первой строки в области объекта Range.
Font	Возвращает объект Font, используемый при написании текста в области объекта Range.
FormatConditions	Возвращает коллекцию условных форматов, содержащую не более трех элементов - объектов класса
FormatCondition.	Объект Range может иметь до трех условных форматов, выбор каждого из которых зависит от выполнения условия форматирования. Условие определяется параметрами объекта FormatCondition - оператором условия и константой, которая сравнивается со значением выражения, заданного объектом Range. В простейшем случае, когда объектRange задает ячейку, то значение в ячейке сравнивается с заданной константой. МетодAdd коллекции позволяет задать новое условие форматирования. Методы Modify иDelete объекта FormatCondition позволяю т модифицировать или удалять существующий формат. Параметры формата задаются с использованием объектовBorders, Font и Interior, возвращаемых свойствами объекта FormatCondition.
Formula, FormulaR1C1, FormulaArray,	Первое из них позволяет прочесть или задать формулу в формате A1, второе - в формате R1C1, третье -формулу над массивами. Остальные также так или иначе связаны с заданием формул.

FormulaLocal, FormulaHidden, FormulaLabel, FormulaR1C1Local	
Locked	Возвращает значение True, если объект закрыт для модификаций и False, если модификация данного объекта возможно, хотя рабочий лист защищен. ВозвращаетсяNull, есди в области объекта Range существуют закрытые и открытые ячейки.
Offset	Об этом свойстве, возвращающем объект Range, у уже подробно рассказывал.
Style	Свойство имеет статус "только для чтения" - возвращает объект Style, характерный для объекта Range.
Value	Значение указанной ячейки. Если она пуста, то возвращается значение Empty, что можно проверить, вызвав функцию IsEmpty. Если объект Range содержит более одной ячейки, то возвращается массив значений, что можно проверить, вызвав функциюIsArray. Функции IsNumber, IsText позволяют определить тип значения, хранимого в ячейке.

Методы объекта Range

Метод	Действие
AutoFit	Автоматически настраивает ширину столбца и высоту строки
Clear, ClearComments, ClearContents, ClearFormats	Метод clear очищает диапазон. В следующем примере очищается диапазон A1:G37. Range("A1:G37").Clear Методы ClearComments, ClearContents, ClearFormats и ClearNotes очищают в указанном диапазоне
Insert	Вставка ячейки или диапазона ячеек. В следующем примере вставляется новая строка перед четвертой строкой рабочего листа Лист1: Worksheets("Лист1").Rows(4).Insert
Select	Выделение диапазона
ClearNotes	Комментарии, содержание, форматы и примечания соответственно
Copy	Копирует диапазон в другой диапазон или в буфер обмена. Синтаксис: Copy(destination) Аргумент destination определяет диапазон, куда копируется данный диапазон. Если аргумент destination опущен, то копирование происходит в буфер обмена. В данном примере диапазон a1:D4 рабочего листа копируется в диапазон E5 листа 2: Worksheets("Лист1").Range("A1:D4").Copy destination:=Worksheets("Лист2").Range("E5")
Delete	Удаляет диапазон
AddComment	Добавляет примечание к диапазону. Синтаксис: AddComment (Text) Text — строковое выражение, добавляемое в качестве примечания
Address	Возвращает адрес ячейки. Синтаксис: Address(rowAbsolute, columnAbsolute, referenceStyle, external, relativeTo) Аргументы: rowAbsolute — допустимы два значения True и False, если используется значение True или аргумент опущен, то возвращается абсолютная ссылка на строку; columnAbsolute — допустимы два значения True и False, если используется значение True или аргумент опущен, то возвращается абсолютная ссылка на столбец;

Метод	Действие
	referenceStyle — допустимы два значения xiAl и xlRld, если используется значение xiAl или аргумент опущен, то возвращается ссылка в виде формата A1; external — допустимы два значения True и False, если используется значение False или аргумент опущен, то возвращается относительная ссылка. Следующий пример показывает различные результаты адресации. MsgBox Cells(1, 1).Address В диалоговом окне отображается адрес A1. MsgBox Cells(1, 1).Address(rowAbsolute:=False В диалоговом окне отображается адрес $A1. MsgBox Cells(1, 1).Address(referenceStyle:=xlR1C1 В диалоговом окне отображается адрес R1C1
Cut	Копирует диапазон с удалением в указанный диапазон или в буфер обмена. Синтаксис: Cut(destination) Аргумент destination определяет диапазон, который копируется в данный диапазон. Если аргумент destination опущен, то диапазон копируется в буфер обмена
Columns, Rows	Возвращают соответственно семейства столбцов и строк, из которых состоит диапазон. В следующем примере переменным i и j присваиваются значения, равные количеству столбцов и строк в выделенном диапазоне соответственно: i = Selection.Columns.Count j == Selection.Rows.Count

Приложение 3

Синтаксис событий элементов управления

- *Activate* – данное событие генерируется, когда форма становится активной
- *AddControl* – событие генерируется, когда в форму, рамку или на вкладку добавляется новый элемент управления.
- *AfterUpdate* – генерируется, когда меняем данные в элементе управления.
- *BeforeDragOver* – генерируется перед инициализацией процесса перемещения (drag-and-drop).
- *BeforeDropOrPaste* – генерируется, когда пользователь переносит или вставляет данные в объект.
- *BeforeUpdate* – генерируется перед обновлением данных в элементе управления.
- Click – генерируется при щелчке мыши по элементу правления.
- *DbClick* – генерируется при двойном щелчке по элементу управления.
- *Deactivate* – генерируется, когда форма становится неактивной.
- *DropButtonClick* – генерируется при каждом раскрытии или сворачивание раскрывающегося списка.
- *Enter* – генерируется перед получением фокуса от другого элемента управления, находящегося в этой же форме.
- *Exit* – событие генерируется перед потерей элементом управления фокуса.
- *Error* – генерируется при возникновении ошибки.
- *Initialize* – генерируется сразу после загрузке объекта, перед его визуализацией.
- *KeyDown* – генерируется при нажатии на клавишу.
- *KeyPress* – генерируется при нажатии любой клавиши, сочетания клавиш Ctrl+<любая клавиша>, клавиши Backspace, клавиши Esc.
- *KeyUp* – генерируется при отпускании клавиши.
- *Layout* – генерируется при изменении элемента управления текущей формы.
- *MouseDown* – генерируется при нажатии на кнопку мыши.
- *MouseUp* – генерируется, когда клавиша мыши отпускается.
- *MouseMove* – генерируется при движении указателя мыши на объектом, когда объект активный, т.е. находится в фокусе.
- *RemoveControl* – генерируется при удалении объекта из родительского объекта-контейнера.
- *Scroll* – данное событие генерируется при перемещении бегунка полосы прокрутки формы.
- *SpinDown* – событие генерируется при нажатии нижней или левой кнопки элемента управления *SpinButton* (счетчик).
- *SpinUp* – событие генерируется при нажатии верхней или правой кнопки в элементе управления *SpinButton* (счетчик).
- *Terminate* – генерируется при выгрузке объекта из памяти или при удалении связи между объектной переменной и областью памяти, где расположен объект.
- *Zoom* – генерируется при изменении масштаба.

Рассмотрим некоторые из свойств элементов управления.

- *Объект*.***Accelerator*** [=String] – это свойство позволяет получить или назначить *Объекту* комбинацию клавиш Alt+<Клавиша>.
- *Объект*.***ActiveControl*** – показывает что элемент управления *Объект* становится активным.

– *Объект*.***Alignment*** [=*fmAlignment*] – выравнивает заголовок элемента управления, указанного в свойстве *Caption*. Параметр *fmAlignment* принимает оно из следующих значений: *fmAlignmentLeft* – заголовок находится левее элемента управления, *fmAlignmentRight* – заголовок находится правее элемента управления.

– *Объект*.***AutoSize*** [=**Boolean**] – автоматическое масштабирование размера надписи в зависимости от высоты объектов. Если свойство *AutoSize* имеет значение **True**, то размер шрифта подбирается автоматически.

– *Объект*.***AutoTab*** [=**Boolean**] – свойство позволяют указать, должен ли осуществляться переход фокуса к следующему по табулированному порядку элементу управления при максимальном заполнении текстового поля.

Объект.***AutoWordSelect*** [=**Boolean**] – это свойство задает базисную единицу при распространении выделения. Если свойство имеет значение **True**, то базисной единицей является полное слово. Если **False**, то один символ.

– *Объект*.***BackColor*** [=**Long**] – задает цвет фона объекта.

– *Объект*.***BackStyle*** [=*fmBackStyle*] – задает стиль отображения фона объекта. Параметр *fmBackStyle* имеет следующие значения: *fmBackStyleTransparent* – прозрачная основа, *fmBackStyleOpaque* – непрозрачная.

– *Объект*.***BorderColor*** [=**Long**] – задает цвет рамки, ограничивающей объект.

– *Объект*.***BorderStyle*** [= *fmBorderStyle*] – свойство задает ограничивающую рамку. Параметр fmBorderStyle имеет следующие значения: *fmBorderStyleNone* – рамка отсутствует, *fmBorderStyleSingle* – рамка есть толщиной в один пиксель.

– *Объект*.***BoundColumn*** [=**Variant**] – задает столбец, значение которого будет отображаться в свойстве **Value**. Это свойство есть у следующих элементов управления: **ComboBox**, **ListBox** они могут иметь несколько столбцов.

– *Объект*. ***BoundValue*** [=**Variant**] – свойство указывает на состояние объекта, когда он получает фокус. Параметр **Variant** для разных элементов управления имеет разные значения. Для элементов управления **Checkbox**, **OptionButton**, **ToggleButton** – **Null**, когда элемент активен, но не доступен; -1(**True**) – элемент выбран; 0(**False**) – элемент не выбран. Для элементов **ScrollBar**, **SpinButton** – целочисленное значение, находящееся в интервале между *Max* и *Min* (указано в этих свойствах) и отражающее текущее положение. Для элементов **ComboBox**, **ListBox** – значение свойства *BoundColumn* для выбранной строки. Для **CommandButton** – значение всегда **False**. Для *MultiPage* – целочисленное значение, показывающее индекс текущей активной вкладки. Первая вкладка имеет индекс 0, вторая – 2 и т.д. Для **TextBox** – строковое выражение, набранное в текстовом поле.

– *Объект*.***Cancel*** [=**Boolean**] – для командной кнопки *Объект* показывает, что она является кнопкой Cancel для формы. Это свойство имеет значение **True** только для одного объекта формы.

– *Объект*.***CanPaste*** – позволяет узнать, может *Объект* получить данные из буфера обмена. В том случае если значение **True**, то вставка из буфера возможна.

– *Объект*.***Caption*** [=**String**] – содержит текст, находящийся в поле надписи.

– *Объект*.***CanRedo*** – это свойство проверяет можно ли для объекта *Объект* выполнить возврат к последнему отмененному действию. Если данное свойство имеет значение **True**, то можно вернуться.

– *Объект*.***CanUndo*** – определяет возможность отмены последнего действия. Если данное свойство имеет значение **True**, то отмена возможна.

– *Объект*.***ColumnCount*** [=**Long**] – свойства содержит или позволяет установить число столбцов в элементах управления **ComboBox**, **ListBox**.

– *Объект*.***ControlTipText*** [=**String**] – содержит текст всплывающей подсказки, которая появляется, когда указатель мыши попадает в фокус объекта.
– *Объект*.***Count*** – указывает число объектов, находящихся в семействе объекта *Объект*.
– *Объект*.***CurLine*** [=**Long**] – содержит значение текущей строки, где в данный момент находится курсор. Первая строка имеет индекс 0, вторая – 1, третья – 2 и т.д.
– *Объект*.***Default*** [=**Boolean**] – показывает на командную кнопку, которая при инициализации формы по умолчанию будет иметь фокус.
– *Объект*.***Delay*** [=**Long**] – задает время задержки пере генерацией событий: *SpinUp*, *SpinDown*, *Change* (в миллисекундах), для элементов управления **SpinButton** и **ScrollBar**.
– *Объект*.***DragBehavior*** [=*fmDragBehavior*] – дает возможность использовать «перетащить – и – оставить» в полях редактирования элементов управления **TextBox** и **ComboBox**. Значение параметра *fmDragBehavior*: если *fmDragBehaviorDisabled*, то перетаскивание невозможно, если *fmDragBehaviorEnabled*, то выделенные фрагменты можно перетаскивать в пределах поля редактирования.
– *Объект*.***Enabled*** [=**Boolean**] – позволяет делать *Объект* недоступным, т.е. *Объект* приобретает серый цвет и не реагирует на любые действия над ним.
– *Объект*.***ForeColor*** [=**Long**] – определяет цвет шрифта элемента управления.
– *Объект*.***GroupName*** [=**String**] – позволяет задать имя группы, объединяющей несколько элементов управления **OptionButton**.
– *Объект*.***Index*** [=**Integer**] – показывает значение, характеризующее положение вкладок (объектов **Tab** и **Page**) друг относительно друга в семействах **Tabs** и **Pages**. Первая вкладка имеет индекс – 0, вторая – 1 и т.д.
– *Объект*.***LayoutEffect*** – позволяет узнать, был ли перемещен элемент управления при изменении положения. Свойство может принимать два значения: *fmLayoutEffectNone* – элемент управления не был перемещен, *fmLayoutEffectInitiate* – был перемещен.
– *Объект*.***LineCount*** – возвращает число текстовых строк, определенных для элементов управления **TextBox** и **ComboBox**.
– *Объект*.***ListCount*** – возвращает число строк, имеющихся в списке.
– *Объект*.***ListIndex*** [=**Variant**] – указывает на номер строки, где находится курсор в элементах управления **ListBox**, **ComboBox**. Имеет значение 0 – выбрана первая строка, 1 – выбрана вторая строка и т.д.
– *Объект*.***Locked*** [=**Boolean**] – показывает, может или нет элемент управления быть отредактирован. Если свойство имеет значение **True** – никакие изменения внести невозможно. Если **False** – возможно редактирования объекта.
– *Объект*.***MatchEntry*** [=*fmMatchEntry*] – устанавливает порядок поиска в списках элементов управления **ListBox** и **ComboBox**. Параметр *fmMatchEntry* может принимать следующие значения: *fmMatchEntryFirstLetter* – производится поиск по первому введенному символу. После ввода первого символа в списке выполняется автоматический поиск и выделение слов, начинающихся с этого символа. Для перехода к следующему слову, начинающегося с этого символа, введите его снова. Поиск происходит циклически. Если *fmMatchEntryComplete* – поиск производится по совпадению последовательно введенных символов. Если *fmMatchEntryNone* – поиск не производится.
– *Объект*.***MatchFound*** – это свойство устанавливает, соответствует ли текст, введенный пользователем, одному из элементов, имеющихся в списке. Если **True** – соответствие найдено. Если **False** – соответствие не обнаружено.
– *Объект*.***MatchRequired*** [=**Boolean**] – показывает, должен ли текст, введенный в область редактирования, обязательно совпадать с одной из имеющихся строк списка.

Если **True** – до тех пор, пока не будет достигнуто соответствие между введенным и имеющимся текстом,передача фокуса другому элементу управления невозможно. Если **False** – позволяет вводить в область редактирования текст, не совпадающий со списком.

- *Объект*.***MultiLine*** [=**Boolean**] – показывает может ли элемент управления работать с несколькими строками (**TextBox**).
- *Объект*.***PasswordChar*** [=**String**] – позволяет скрыть текст, вводимый в поле, символами – заменителями. Параметр **String** содержит тот символ, на который будут заменяться вводимый текст.
- *Объект*.***RowSource*** [=**String**] –содержит ссылку на источник данных для списков элементов управления **ListBox** и **ComboBox**.
- *Объект*.***Selected***(*index*) [=**Boolean**] – определяет состояние элемента списка, выделен он или нет. Если свойство имеет значение **True** – элемент с индексом *index* выделен.
- *Объект*.***SelectedItem*** – возвращает ссылку на текущий выделенный объект **Tab** или **Page**.
- *Объект*.***SelLength*** [=**Long**] – содержит число символов, выделенных в текстовом поле **TextBox** или поле редактирования элемента управления **ComboBox**.
- *Объект*.***SelText*** [=**String**] – содержит текст, который в данный момент является выделенным.
- *Объект*.***TabIndex*** [=**Integer**] – содержит индивидуальный индекс объекта, определяющий последовательность обхода объектов в форме.
- *Объект*.***Tag*** [=**String**] – свободное свойство, которое может быть использовано пользователем по своему усмотрению.
- *Объект*.***Text*** [=**String**] – содержит строковое выражение, отображаемое в поле элемента управления **TextBox** или в выбранной строке элементов управления **ComboBox** или **ListBox**.
- *Объект*.***Value*** [=**Variant**] – содержит значение, указывающее на текущее состояние элемента управления. Параметр **Variant** для разных элементов управления имеет разные значения. Для элементов управления **Checkbox**, **OptionButton**, **ToggleButton** – **Null**, когда элемент активен, но не доступен; -1(**True**) – элемент выбран; 0(**False**) – элемент не выбран. Для элементов **ScrollBar**, **SpinButton** – целочисленное значение, находящееся в интервале между *Max* и *Min* (указано в этих свойствах) и отражающее текущее положение. Для элементов **ComboBox**, **ListBox** – значение свойства *BoundColumn* для выбранной строки. Для **CommandButton** – значение всегда **False**. Для **MultiPage** – целочисленное значение, показывающее индекс текущей активной вкладки. Первая вкладка имеет индекс 0, вторая – 2 и т.д. Для **TextBox** – строковое выражение, набранное в текстовом поле.
- *Объект*.***Visible*** [=**Boolean**] – определяет, является ли элемент управления видимым или нет. Если **True** – элемент управления видим, если **False** – не видим.

Здесь представлены некоторые из методов, наиболее используемые.

- *Объект*.***Add*** – метод **Add** имеет два основных синтаксиса:
- **Set** *Object* = *Объект*.**Add**([**Name**[, **Caption**[, **index**]]]) – необходимо для добавления объектов Tab и Page в элементы управления **TabStrip** и **MultiPage**.
- **Set** *Control* = *Объект*. **Add**(**ProgID**[, **Name**[, **Visible**]]) – для всех остальных элементов управления.
- **Variant** = *Объект*.**AddItem**[**Item**[, **varIndex**]] – добавляет в списки, состоящие из одного столбца, новый элемент, а в списки, состоящие из нескольких столбцов, новую строку. Параметр **Item** – идентификатор элемента или строки, **varIndex** – указывает на идентификатор элемента, за которым будет размещен новый элемент или строка.

- *Объект*.***Clear*** – удаляет все объекты из объекта или семейства *Объект*.
- *Объект*.***Copy*** – копирует содержимое объекта *Объект* в буфер обмена.
- *Объект*.***Cut*** – копирует выделенное содержимое объекта *Объект* в буфер обмена. После этого данные удаляются.
- **Set** *Object* = *Объект*.**Item**(*collectionindex*) – этот метод позволяет получить объект семейства по его индексу в семействе. Параметр *collectionindex* – включает в себя имя или индекс объекта, содержащегося в семействе *Объект*.
- *Объект*.***Paste*** – служит для вставки в объект *Объект* данных из буфера обмена.
- **Boolean** = *Объект*.***RedoAction*** – восстанавливает последнее отмененное действие. Отмена последнего выполненного действия осуществляется с помощью метода **UndoAction**.
- *Объект*.***Remove***(*collectionindex*) – удаляет объект *Объект* из семейства, формы, рамки или вкладки. Параметр *collectionindex* – строковое или целочисленное значение, определяющее имя или индекс объекта. Индекс надо указывать, если объект удаляется из семейства. Этот метод позволяет удалять только те объекты, которые были созданы динамически во время выполнения программы.
- **Boolean** = *Объект*.***RemoveItem***(*index*) – удаляет из списка строку с индексом *index*.
- *Объект*.***SetFocus*** – этот метод позволяет передать фокус объекту *Объект*.

Приложение 4

Определение первой заполненной ячейки на листе

```
Sub Get_First_Cell()
 Dim FirstRow As Long, lFirstCol As Long, rFndRng As Range
'проверяем, есть ли данные в первой ячейке диапазона данных
If ActiveSheet.UsedRange.Cells(1, 1) <> "" Then
FirstRow = ActiveSheet.UsedRange.Row
FirstCol = ActiveSheet.UsedRange.Column
Else
'ищем ячейку с любым значение(так же с формулой)
Set rFndRng = ActiveSheet.UsedRange.Find("*", , xlFormulas, xlWhole)
If rFndRng Is Nothing Then
MsgBox "Лист не содержит данных", vbInformation, "Информация": Exit Sub
End If
lFirstRow = rFndRng.Row: lFirstCol = rFndRng.Column
End If
MsgBox "Номер строки первой заполненной ячейки: " & lFirstRow & vbNewLine & _
 "Номер столбца первой заполненной ячейки: " & lFirstCol
End Sub
```

Учебное текстовое электронное издание

Калугина Ольга Борисовна
Надеина Маргарита Владимировна
Лукьянов Георгий Игоревич

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Учебное пособие

1,19 Мб
1 электрон. опт. диск

г. Магнитогорск, 2015 год
ФГБОУ ВПО «МГТУ»
Адрес: 455000, Россия, Челябинская область, г. Магнитогорск,
пр. Ленина 38

ФГБОУ ВПО «Магнитогорский государственный
технический университет им. Г.И. Носова»
Кафедра информатики и информационной безопасности
Центр электронных образовательных ресурсов и
дистанционных образовательных технологий
e-mail: ceor_dot@mail.ru