

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

**И.С. Осетрова, Н.А. Осипов**

**Microsoft Visual Basic for Application**

**Учебное пособие**



**Санкт-Петербург**

**2013**

УДК 004.655, 004.657, 004.62

И.С. Осетрова, Н. А. Осипов

Microsoft Visual Basic for Application - СПб: НИУ ИТМО, 2013. – 120 с.

В пособии представлено руководство по программированию на Visual Basic for Application в MS Office по дисциплине «Создание клиентских приложений».

Предназначено для студентов, обучающихся по всем профилям подготовки бакалавров направления: 210700 Инфокоммуникационные технологии и системы связи.

Рекомендовано к печати Ученым советом факультета инфокоммуникационных технологий, протокол №3 от 19 марта 2013 г.



В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на **2009–2018** годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

© Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, 2013

© И.С. Осетрова, , Н. А. Осипов 2013.

# Оглавление

<b>Введение.....</b>	<b>5</b>
<b>1. Проектирование пользовательского приложения.....</b>	<b>6</b>
1.1. Microsoft Office 2010 – среда разработки.....	6
1.2. VBA - язык офисного программирования .....	6
1.3. Microsoft Office 2010 - система приложений.....	7
1.4. Особенности проектирования .....	8
<b>2. Использование редактора Visual Basic .....</b>	<b>9</b>
2.1. Место хранения кода.....	9
2.2. Работа в редакторе Visual Basic .....	10
2.3. Автоматическая запись макросов .....	16
2.4. Написание программного кода .....	17
2.5. Запуск кода .....	18
2.6. Перемещение компонентов .....	19
2.7. Практическое занятие 1: Использование макрорекодера.....	19
<b>3. Использование Visual Basic for Application .....</b>	<b>25</b>
3.1. Основы синтаксиса.....	25
3.2. Переменные, типы данных, константы .....	25
3.3. Процедуры.....	29
3.4. Управляющие инструкции.....	32
3.5. Обработка ошибок.....	35
3.6. Отладка (Debugging).....	40
3.7. Практическое занятие 2: Использование Visual Basic for Applications	
41	
<b>4. Формы и объекты управления.....</b>	<b>47</b>
4.1. Введение: элементы управления в формах и документах.....	47
4.2. Использование элементов управления в формах .....	48
4.3. Работа с формой.....	50
4.4. Использование элементов управления в документах .....	53
4.5. Практическое занятие 3: Использование элементов управления.....	54
<b>5. Объектные модели и автоматизация.....</b>	<b>61</b>
5.1. Введение. Объекты, классы, объектные модели .....	61
5.2. Объектная иерархия объектов приложения.....	62
5.3. Работа с объектами.....	64
5.4. Автоматизация офисных приложений .....	65
5.5. Практическое занятие 4: Модели объектов и автоматизация.....	69
<b>6. Использование Microsoft Excel объектов .....</b>	<b>74</b>
6.1. Объектная модель Microsoft Excel.....	74
6.2. Использование объекта Workbook и коллекции Workbooks.....	75

6.3. Использование Worksheet.....	76
6.4. Создание Charts (диаграмм) .....	77
6.5. Создание сводных таблиц .....	80
6.6. Практическое занятие 5: Работа с Microsoft Excel .....	82
<b>7. Использование Microsoft Word объектов.....</b>	<b>87</b>
7.1. Объектная модель Microsoft Word .....	87
7.2. Работа с Word документами.....	88
7.3. Работа с частями (областями) документа .....	89
7.4. Работа с текстом .....	91
7.5. Практическое занятие 6: Работа с Microsoft Word .....	93
<b>8. Работа с базами данных .....</b>	<b>99</b>
8.1. Обзор технологии ADO .....	99
8.2. Коллекция Fields и объекты Field.....	105
8.3. Сортировка и фильтрация данных .....	107
8.4. Объект Command и коллекция Parameters.....	108
8.5. Практическое занятие 7: Работа с базой данных .....	111
<b>Литература.....</b>	<b>115</b>

## **Введение**

Курс предназначен для студентов, обучающихся по всем профилям подготовки бакалавров направления: 210700 «Инфокоммуникационные технологии и системы связи».

В результате курса, проводимого под руководством преподавателя, студенты смогут:

- использовать язык программирования Microsoft VBA для написания программ;
- использовать редактор Visual Basic для создания и отладки программ;
- использовать объектные модели MS Office 2007/2010;
- создавать формы и элементы управления для ввода данных и запуска макросов;
- программировать в MS Office Word 2007/2010;
- программировать в MS Office Excel 2007/2010;
- программировать в MS Office Access 2007/2010;
- обращаться из приложений Office к базам данных, используя объектную модель ADO.

Для прохождения данного курса студенты должны иметь:

- базовые знания по архитектуре персонального компьютера;
- базовые знания операционной системы Microsoft Windows и ее основных возможностей;
- практический опыт работы в MS Office 2007/2010, (как минимум грамотное владение приложениями MS Office Word и MS Office Excel);
- умение формализовать решаемую задачу (наличие алгоритмического мышления).

# 1. Проектирование пользовательского приложения

## 1.1. Microsoft Office 2010 – среда разработки

Microsoft Office 2010 – система приложений, объединенных единым интерфейсом и возможностью переноса данных. Это также платформа для создания собственных приложений.

Каждое из приложений Microsoft Office 2010 имеет два уровня использования:

- Применение приложения как готового универсального инструмента
- Применение приложения как инструментальной среды для разработки собственных приложений, решающих специфические задачи.

Microsoft Office 2010 – масштабируемая среда разработки. Это означает, что она подходит как для решения небольших, простых задач, так и для решения сложных. К *простым* задачам относятся создание отдельных документов или создание сравнительно небольшой системы документов. *Сложными* задачами могут быть, например:

- разработка клиентской части клиент серверного приложения, рассчитанных на работу большого числа пользователей с серьезной базой данных, такой как Microsoft SQL Server или Oracle.
- разработка документов для совместного использования в Интернете.
- разработка большой системы документов, обеспечивающих «полную» автоматизацию деятельности офиса.

Microsoft Office содержит среду разработки - редактор Visual Basic (Visual Basic Editor) в отдельном от основного приложения окне.

Office 2010 обладает мощным языком программирования - Visual Basic for Applications (VBA), общим для приложений Microsoft Office 2010. Visual Basic for Applications представляет из себя расширение языка Visual Basic 6.0 объектами, определяющими Office 2010.

## 1.2. VBA - язык офисного программирования

VBA является современным языком визуального и объектно-ориентированного программирования, который позволяет создавать собственные объекты и работать с огромным числом объектов, содержащихся в библиотеках. Все приложения, входящие в состав Office 2010 представляют собой совокупность объектов со своими свойствами, методами и событиями.

С помощью VBA можно создавать комплексные приложения, одновременно использующие те или иные компоненты приложения.

Office 2010 содержит большое количество программируемых объектов, функциональные возможности которых Вы можете использовать при разработке своего пользовательского приложения.

**Основной задачей** офисного программирования является создание документа.

В данном контексте "**документ**" понимается как **объект** в объектно-ориентированном программировании и представляет собрание данных разного типа и программ, обрабатывающих эти данные.

Под документами Office понимают документы разных типов - рабочие книги Excel, документы Word, базы данных Access и презентации PowerPoint. С любым из этих документов связываются и данные и программы.

Все создаваемые программные компоненты документа объединяются в одно целое, называемое **проектом**:

- проект является частью документа и не существует вне его,
- проект хранится вместе с документом, его невозможно отделить от документа,
- невозможно создать независимый от документа проект на VBA.

### ***1.3. Microsoft Office 2010 - система приложений***

Основные возможности всех приложений Office 2010

- Объектная модель всех приложений
- Использование шаблонов – файлов, на базе которых можно создать семейство близких документов. Шаблоны задают общие свойства, и требуется лишь сравнительно небольшая настройка для получения конкретного документа.
- Использование надстроек —дополнительных программ, расширяющих возможности приложения путем добавления специальных команд и новых функций.
- Макрорекодер – транслятор действий пользователя на язык программирования. Результатом действия макрорекодера является создание макроса (процедуры без параметров).
- Использование ActiveX (дополнительных элементов управления, расширяющих возможности взаимодействия приложения и пользователя)
- Мастера – инструмент пошагового процесса достижения цели.
- Использование объектных моделей Office 2010

#### **Microsoft Excel**

*Microsoft Excel* – электронные таблицы, позволяющие производить разнообразные вычисления над данными, хранить их, просматривать в виде таблиц, графиков, диаграмм.

Некоторые возможности Microsoft Excel:

- **События** *Основные объекты* Microsoft Excel – workbooks и worksheets обладают событиями, используя которые можно получить контроль над объектами.
- **Сводные таблицы** (PivotTables) – сводные таблицы можно создавать и изменять программно.
- **Проверка данных** – есть возможность проверять данные при вводе (без использования программного кода).

- **Использование шаблонов** - Шаблоном называется книга, имеющая определенное содержимое и элементы форматирования, и используемая как образец при создании новых книг Excel. Шаблоны можно создавать как для книг, так и для отдельных листов.

Шаблон может включать в себя стандартный текст, например заголовки страниц, подписи строк и столбцов, макросы Visual Basic, а также нестандартные панели инструментов.

#### **Microsoft Word**

*Microsoft Word* – мощный текстовый редактор, предназначенный для создания профессиональных документов и для упрощения задач обработки текстов.

Некоторые возможности Microsoft Word:

- **События** *Основные объекты* Microsoft Word – document (документ) и template (шаблон) обладают событиями, используя которые можно получить контроль над объектами.
- **Использование шаблонов** - Вы можете сохранять макросы в документах и шаблонах. Шаблон - Особый вид документа, предоставляющий специальные средства для оформления итогового документа. Шаблон может содержать следующие элементы:
  - текст или формат, одинаковый для всех документов этого типа, например, для служебной записки;
  - стили;
  - элементы автотекста; элементы автозамены;
  - макросы;
  - меню и присвоенные сочетаниям клавиш операции;
  - панели инструментов.

#### **1.4. Особенности проектирования**

Вы можете хранить свой программный код в шаблоне, в документе или в надстройках.

Надстройка - вспомогательный компонент, который можно загрузить вместе с каким-либо приложением для повышения функциональных возможностей приложения.

Приложения Microsoft Office 2007/10 по умолчанию не позволяют запускать макросы. Поэтому перед тем, как приступить к созданию макросов, необходимо открыть окно настройки **Параметры Excel** (или другого приложения), затем щелкнуть по строке **Центр управления безопасностью**, затем по кнопке параметры **Центра управления безопасностью**. В открывшемся окне **Центр управления безопасностью** необходимо перейти на строку **Макросы** и установить переключатель в положение *Включить все макросы*. Рекомендуется также установить флажок *Доверять доступ к объектной модели приложений VBA*. Затем нужно закрыть и открыть приложение.



## 2. Использование редактора Visual Basic

### 2.1. Место хранения кода

#### Связь проекта и документа

- **Проектом** в Office 2010 называется набор программных модулей, связанных с основным документом приложения. Программный проект – часть документа, которая хранится вместе с документом и не может быть отделена от него.
- Каждому документу (или шаблону) соответствует свой проект. Каждый проект связан со своим документом (шаблоном).

#### Код в Microsoft Excel

В Microsoft Excel программный код может храниться в:

- **Рабочих книгах (workbooks)** - в текущей или личной книге макросов. Код сохраняется с рабочей книгой и становится доступным, когда пользователь открывает ее. По умолчанию Excel помещает макрос в модуль активной рабочей книги. При желании можно записать его либо в новой рабочей книге (откроется пустая новая книга) либо в личной книге макросов. Это рабочая книга называется Personal.xlsb и хранится в папке XLStart. Загрузка книги происходит после запуска Excel и, соответственно, код доступен всегда при работе с Microsoft Excel. Данная книга скрыта, поэтому при обычной работе не видна. Файл Personal.xlsb не существует до тех пор, пока не будет записан в нее первый макрос.
- **Шаблонах (templates)** Код может сохраняться в шаблоне Microsoft Excel. Сохраняя рабочую книгу как шаблон, Вы затем можете создавать новые рабочие книги на базе этого шаблона. Когда Вы создаете книгу на базе шаблона, Microsoft Excel копирует шаблон в рабочую книгу. Код, который находится в шаблоне, также копируется. В отличие от документа Word, который связан со своим шаблоном, Microsoft Excel не связан с шаблоном, т.е. любые изменения, сделанные в шаблоне, не отразятся на рабочих книгах, ранее созданных на этом шаблоне.
- **Надстройках (add-ins)** Сохраняя рабочую книгу как надстройку, Вы сохраняете ее с расширением .xlam. Затем для расширения возможностей Microsoft Excel, вы можете подгружать созданные надстройки.

#### Код в Microsoft Word

В Microsoft Word программный код может находиться в:

- **Документах (Documents)** Код сохраняется с документом и становится доступным, когда пользователь открывает его.

➤ Шаблонах (templates)

Код может сохраняться в шаблоне Microsoft Word. Сохраняя документ как шаблон, Вы затем можете создавать новые документы на базе этого шаблона.

Документ Word связан со своим шаблоном, поэтому каждый раз, когда Вы открываете документ, созданный на базе шаблона, Microsoft Word загружает связанный шаблон. Если Вы изменяете код в шаблоне документа, измененный код доступен всем документам, созданным на базе этого шаблона.

➤ Normal.dotm

Normal.dotm – стандартный шаблон, используемый всеми документами Microsoft Word. Этот шаблон запускается всегда при старте Microsoft Word. Если Вы хотите, чтобы Ваш код был доступен во всех документах Microsoft Word, сохраняйте его в шаблон Normal. dotm.

## 2.2. Работа в редакторе Visual Basic

### Особенности редактора Visual Basic

Редактор Visual Basic (Visual Basic Editor) – среда разработки проекта – представлена на рис.2.1. Здесь Вы создаете, редактируете, запускаете свой код, написанный для документов Office 2010. Такую среду разработки имеют основные приложения Office 2010 - Microsoft Excel, PowerPoint, Microsoft Word и Microsoft Access.

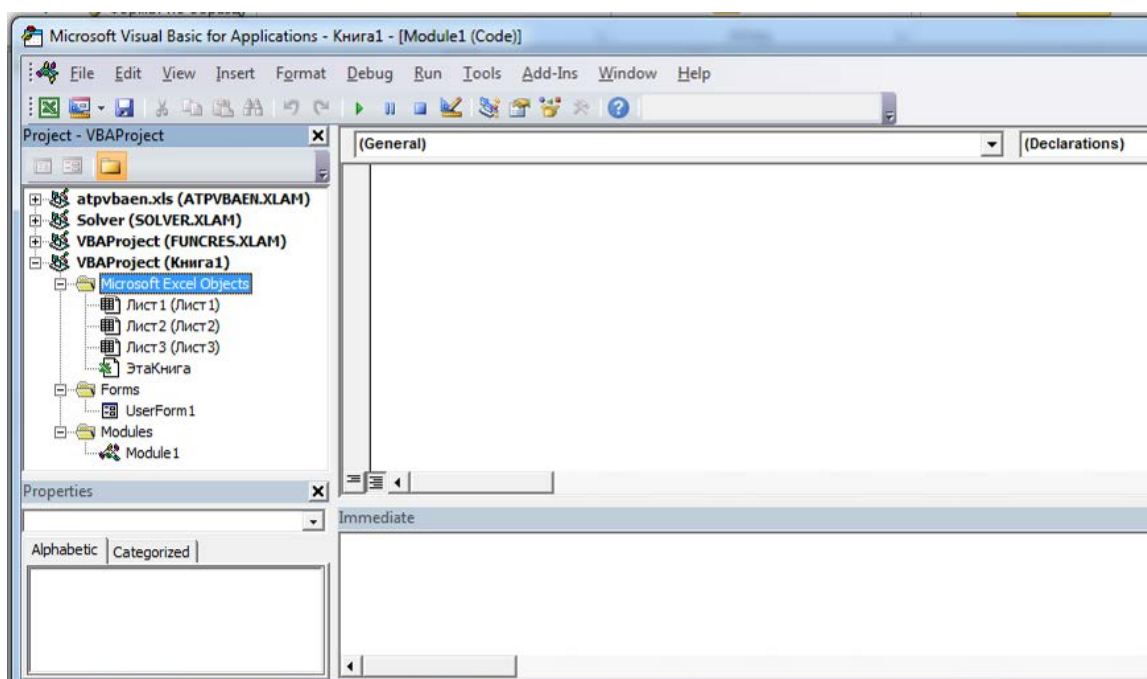


Рис 2.1. Редактор Visual Basic

Каждому объекту в проекте соответствует свое окно кода, такими объектами могут быть:

- сама рабочая книга (ЭтаКнига);
- рабочий лист (например, Лист1);
- модуль VBA (Module1);

- модуль класса (специальный тип модуля, позволяющий создавать новые классы объектов);
- форма UserForm1.

Особенности редактора VBA:

- удобная среда разработки (аналогичная Microsoft Visual Basic)
- одинаковая среда для основных приложений Office 2010
- отдельное от основного приложения окно проектирования (multiple document interface (MDI))

#### Открытие редактора Visual Basic

Открыть редактор можно различными способами (во всех приложениях Office это делается одинаково):

- на специальной вкладке **Разработчик** в группе **Код** выбрать **Visual Basic**; вкладка **Разработчик** содержит инструменты разработки для работы с VBA, элементы управления, которые можно добавлять в документ и т.д. По умолчанию эта вкладка скрыта. Для ее отображения:
  - в Word/Excel 2007:
    - нажать кнопку Microsoft Office, в появившемся окне нажать на кнопку **Параметры Word**,
    - в окне **Параметры Word** установить галочку в поле *Показывать вкладку Разработчик на ленте*.
  - в Word/Excel 2010:
    - на вкладке **Файл** выбрать **Параметры**, чтобы открыть диалоговое окно Параметры,
    - щелкнуть кнопку **Настройка ленты** в левой части диалогового окна,
    - в разделе **Выбрать команды из**, расположенном слева в окне, выбрать **Популярные команды**.
    - в разделе **Настройка ленты**, который находится справа в диалоговом окне, выбрать **Основные вкладки** в раскрывающемся списке, а затем установить флажок *Разработчик*.
- нажать клавиши <Alt>+<F11>;
- вызвать редактор при возникновении ошибки в макросе;
- открыть готовый макрос для редактирования в диалоговом окне **Макрос**.

В редакторе Visual Basic предусмотрены следующие рабочие окна:

**Project Explorer** — окно проводника проекта (см. рис. 2.2). По умолчанию оно открыто и находится в левой части окна редактора Visual Basic. В нем можно просмотреть компоненты проекта и выполнить множество операций. Каждая рабочая книга и открытые в данный момент надстройки рассматриваются как проекты. Проект можно считать коллекцией объектов, организованных в виде иерархической структуры. Проекты можно разворачивать для работы и сворачивать, если они не нужны.

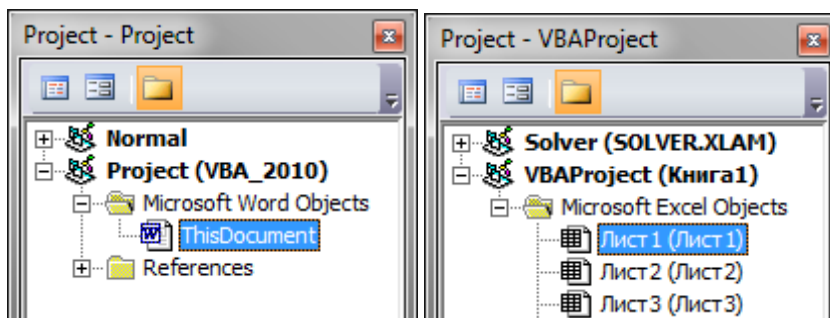


Рис 2.2. Примеры окон проектов в различных приложениях

**UserForm** — окно формы. Появляется тогда, когда вы редактируете пользовательскую форму при помощи дизайнера форм;

**Toolbox** — панель инструментов управления. Из нее можно добавить элементы управления в форму или в сам документ;

**Properties** — одно из самых важных окон. Через него можно просмотреть свойства элемента управления или компонента проекта и изменить их;

**Code** — окно программного кода. В этом окне выполняется основная работа по написанию кода макроса. При открытии программного модуля открывается автоматически;

**Object Browser** — обозреватель объектов. Необходим для получения информации о классах, доступных программе;

**Watch** — окно контролируемых выражений. Используется во время отладки для отслеживания значений выбранных переменных программы и выражений;

**Locals** — окно локальных переменных. Нужно для отслеживания во время отладки значений переменных текущей процедуры;

**Immediate** — окно для немедленного выполнения команд в ходе отладки. Оно позволяет выполнить отдельные строки программного кода и немедленно получить результат.

Найти какое-либо окно можно очень просто: нужно выбрать в меню **View** одноименную команду, и если окно было скрыто, оно появится в редакторе.

#### Основные секции окна проекта:

- Объекты приложения (Microsoft Word Objects, Microsoft Excel Objects)
- Формы (Forms),
- Модули (Modules),
- Модули класса (Class Modules),
- Ссылки (References) – взаимные связи проектов отдельных документов.

#### Application Objects

Объекты приложения (application objects) – секция в окне проекта, содержащая сам документ и объекты его составляющие. Например, в проекте Microsoft Excel – это книга (Microsoft Excel workbook) и листы

(worksheets) этой книги, для Word – сам документ Word, для Access – формы и отчеты, содержащие код.

### Forms

В Microsoft Excel, Word, и PowerPoint Вы можете создавать формы (диалоговые окна для интерактивной работы с пользователем), содержащие элементы управления и ActiveX элементы.

### Modules u Classes

Модули содержат программный код, сопровождающий проект.

Модули класса служат для создания собственных объектов. Модуль класса - контейнер для описания собственного класса.

### References

Вы можете устанавливать ссылки на другие документы, шаблоны и надстройки (того же типа). Например, в Microsoft Word, Вы можете установить ссылку на шаблон.

Для вставки нового компонента в проект, воспользуйтесь пунктом меню **Вставка (Insert)**, а затем выберите необходимый компонент.

Для создания ссылки на другой **Office** документ

1. В меню **Tools** редактора Visual Basic, выберите **References**.
2. В окне **References**, выберите **Browse**.
3. В выпадающем списке **Тип файла**, выберите соответствующий тип файла.
4. Выберите файл, а затем кликнете **Открыть**.

### Свойства проекта

Проект имеет несколько терминальных свойств, которые можно задать в окне **Project Properties** (Свойства проекта). Оно выводится из контекстного меню окна проекта и выбора соответствующего пункта меню. В окне **Свойства проекта** задаются имя проекта, его описание, указываются файлы справки, константы условной компиляции, задается пароль проекта.

#### **Свойства объекта. Окно свойств**

Свойства - характеристики объекта, определяющие его внешний вид или поведение. Например, цвет или заголовок объекта.

Свойства можно изменять во время проектирования (design time) и во время выполнения (run time).

- Свойства объекта *во время проектирования* устанавливаются в окне свойств (см. рис. 2.3).
- Свойства объекта *во время выполнения (run time)* устанавливаются в программе следующим образом:  
Объект.Свойство = Значение

Пример:

```
UserForm1.Caption = "My New Caption"
```

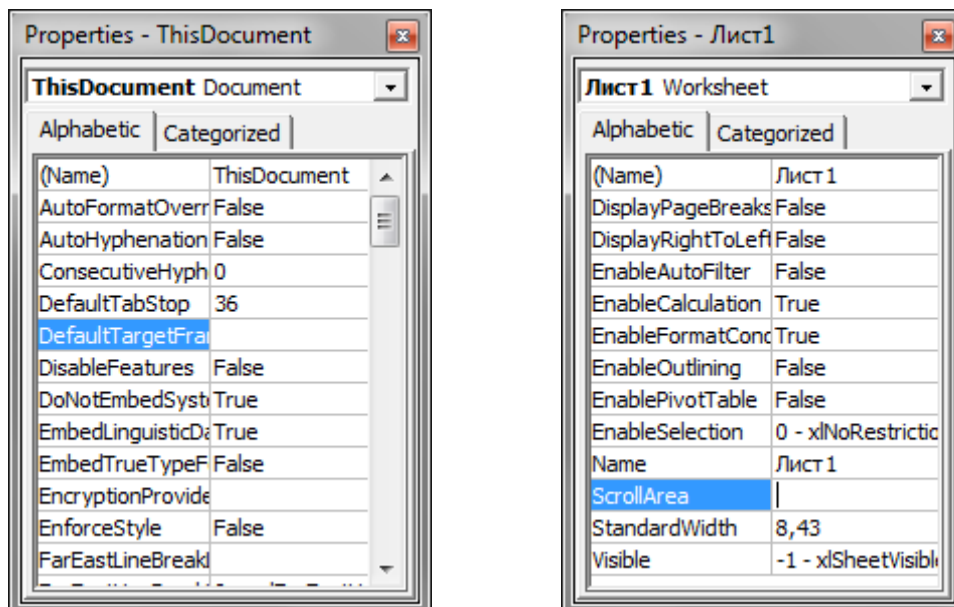


Рис 2.3. Окна свойств объектов

### Окно модуля (программы)

Окно модуля служит для просмотра, написания и редактирования программного кода на Visual Basic for Applications.

Для открытия *Окна модуля* (см. рис. 2.4), надо в *Окне проекта* **выбрать объект** и затем выполнить одно из следующих действий:

- из контекстного меню объекта выбрать соответствующий пункт **View code**;
- нажать функциональную кнопку **F7**;
- выбрать пункт меню **View (Вид)**, команда **Code (Программа)**;
- выполнить двойной щелчок мышью на объекте.

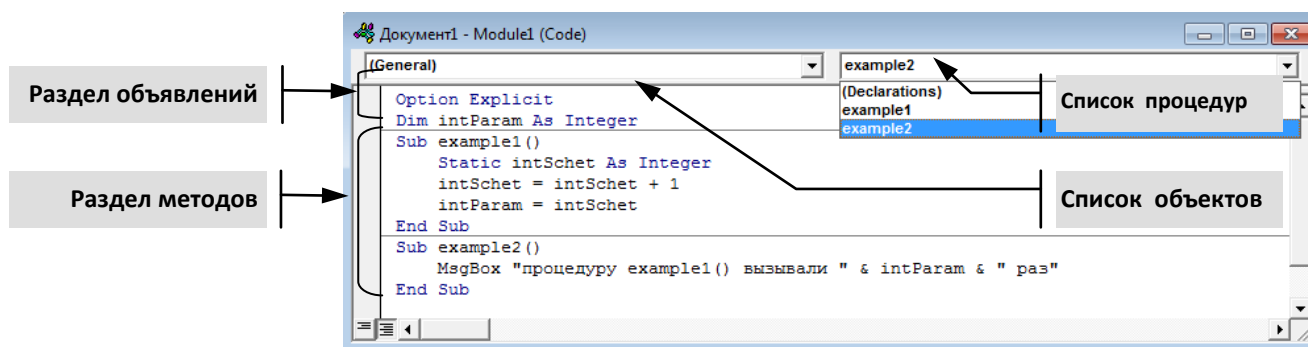


Рис 2.4. Окно модуля

Каждый модуль вне зависимости от его типа (модуль, модуль класса, модуль формы...) имеет всего два раздела:

- Раздел объявлений переменных уровня модуля (General Declarations section). Он идет первым и автоматически отделяется чертой от раздела процедур (методов). Область действия этих переменных распространяется на модуль, но может быть и расширена.
- Раздел процедур (методов) модуля.

В верхней части окна модуля находятся два раскрывающихся списка.

- левый - **Объект** (Object) показывает все объекты выбранного модуля.
- Правый - **Процедура** (Procedure) - список всех возможных событий выбранного объекта.

При выборе события в правом списке (в первый раз), в окне кода автоматически появляется заготовка обработчика этого события. Для стандартных модулей правый список содержит имена всех процедур (методов) стандартного модуля и имя раздела объявлений. Выбор имени из списка позволяет немедленно перейти в соответствующее место окна кода.

#### **Установка параметров редактора**

Для установки параметров редактора Visual Basic, необходимо:

1. В редакторе Visual Basic пункт меню **Tools (Сервис)**, выбрать **Options (Параметры)**.
2. Появится диалоговое окно **Options** с четырьмя вкладками: **Editor, Editor Format, General, Docking**. В диалоговом окне **Options (Параметры)**, установить необходимые параметры и затем кликнуть **ОК**.

#### **Вкладка Editor**

*Параметр Auto Syntax Check* (Автоматическая проверка синтаксиса) определяет, будет ли появляться диалоговое окно при обнаружении редактором ошибки в коде. Рекомендуется параметр включить (дополнительная помощь не мешает), в дальнейшем, при получении опыта программирования можно отключить.

*Параметр Require Variable Declaration* (Обязательное объявление переменных) определяет, будет ли требоваться объявление переменных перед их использованием. При установленном параметре VBE вставляет в начало каждого модуля оператор Option Explicit. В этом случае требуется явно объявить каждую переменную. Помимо экономии ресурсов (если переменная не объявлена, то при использовании она имеет тип Variant – 16 байт) это устраняет опасные ошибки использования похожих имен, например, с (рус) и с (англ).

*Параметр Auto List Members* (Автоматическая вставка объектов) предоставляет помощь при вводе кода, отображая список элементов текущего объекта. Рекомендуется параметр включить.

*Параметр Auto Quick Info* (Отображать краткие сведения) отображает информацию об аргументах функций, свойств и методов, названия которых вводится с клавиатуры.

*Параметр Auto Data Tips* отображает при отладке кода значение переменной, над которой находится указатель мыши.

*Параметр Auto Indent* (Автоматический отступ) определяет автоматическое расположение каждой новой строки с тем же отступом, что и для предыдущей строки.

*Параметр Drag-And-Drop Text Editing* (Редактирование перетаскиванием) позволяет при работе с кодом использовать операцию перемещения текста.

**Параметр Default To Full Module View** (По умолчанию полный режим просмотра) помещает процедуры в одно окно с полосой прокрутки.

**Параметр Procedure Separator** (Разделение процедур) в конце каждой процедуры отображает специальные разделители.

### **Вкладка Editor Format**

Параметры этой вкладки определяют отображение кода, цвет кода, шрифт, размер, полосу индикатора границы (Margin Indicator Bar).

### **Вкладка General**

Содержит общие параметры. Рекомендуется настройки оставить по умолчанию.

### **Вкладка Docking**

Включение параметра для окна означает, что оно прикреплено, т.е. фиксировано по отношению к одной из границ редактора. В результате легче найти требуемое окно, так как оно отображается строго в определенной области.

## **2.3. Автоматическая запись макросов**

Для автоматического создания макросов (процедуры без параметров) используется макрорекордер.

С помощью макрорекордера можно записывать действия, которые пользователь выполняет в программе. Например, записывается ввод и удаление текста, нажатие на кнопки вкладок, форматирование текста и так далее.

Для работы с макрорекордером требуется выполнить следующие действия:

1. На вкладке **Разработчик** нажать кнопку **Запись макроса**, настроить требуемые параметры (имя, место действия).
2. Выполнить действия, которые необходимо автоматизировать.
3. Остановить запись.
4. Использовать записанный макрос по необходимости.

Макрорекордер, помимо написания макросов, поможет вам лучше изучить возможности VBA и тонкости объектных моделей приложений Office. Например, вы не знаете точно, как именно реализовать то, или иное действие программно, но знаете, как это делается вручную. Просто запишите макрос с нужными вам действиями, а потом откройте его в редакторе VBA. Вы сможете проанализировать полученный код, а возможно и включить его фрагменты в свое приложение.

### Абсолютная и относительная адресация

При записи последовательности действий Excel обычно использует *абсолютные* ссылки на ячейки.

В некоторых случаях требуется (например, условия практического задания 1.2) чтобы макрос вводил информацию в активные ячейки. Такой макрос должен быть записан с *относительными* ссылками. Исходная ячейка



выделяется путем вычисления относительного адреса (смещения) и макрос всегда начинает ввод текста в активной ячейке.

Для реализации относительной формы записи следует включить кнопку *Относительные ссылки* в группе **Макросы** на вкладке **Вид** или в группе **Код** на вкладке **Разработчик**. Метод записи можно изменять в любой момент записи макроса.

#### Корректировка макроса, записанного макрорекодером

В окне диалога, которое можно использовать при запуске макроса (вкладка ленты **Вид | Макросы | Макросы...**) кроме кнопки **Выполнить** также содержатся и другие кнопки:

**Изменить** — при нажатии на эту кнопку откроется редактор Visual Basic, который позволяет изменять макросы и создавать собственные программы.

**Создать** — используется для создания нового макроса и открытия редактора для его редактирования.

**Удалить** — для удаления макроса.

В данном случае в окне **Макрос** отображается список макросов из открытого документа. Чтобы посмотреть макросы, находящиеся в других местах (например, в Normal.dotm), следует воспользоваться списком *Макросы из*.

## **2.4. Написание программного кода**

Вариант 1:

1. Открыть в окне проектов необходимый объект с модулем (модуль, форма, документ, рабочий лист...)
2. Перейти в соответствующий модуль и там набрать ключевое слово (**Sub**, **Function** или **Property**), имя процедуры и ее аргументы и затем нажать клавишу Enter, VBA поместит строку с соответствующим закрывающим оператором.
3. Написать текст процедуры между ее заголовком и закрывающим оператором.

Вариант 2:

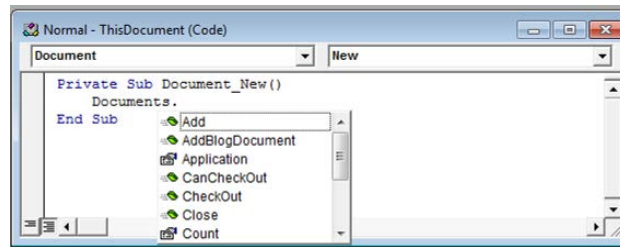
1. Открыть в окне проектов необходимый объект с модулем (модуль, форма, документ, рабочий лист...)
2. Перейти в соответствующий модуль
3. В меню редактора Visual Basic **Вставка (Insert) → Процедура (Procedure)**
4. Заполнить форму «Add Procedure»

Процедуры обработки событий располагаются в модулях, связанных с объектами, реагирующими на события.

#### **Возможности редактора при написании программы**

- Завершение ключевых слов CTRL+ пробел или CTRL+TAB;
- Отображение списка свойств и методов после ввода имени объекта и точки (см. рис. 2.5);

- Появление кратких сведений при написании имени вызываемой процедуры.



*Рис 2.5. Отображение списка свойств и методов*

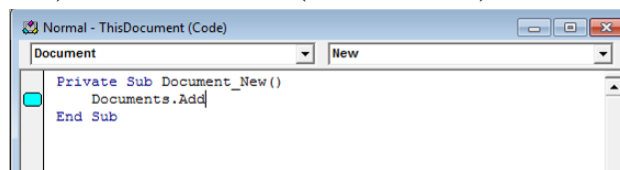
- Переход на текст вызываемой процедуры Shift + F2, обратно - Ctrl + Shift + F2
- Контекстная справка F1.

#### Использование закладок

Для удобства написания и навигации по коду используются закладки (см. рис. 2.6).

*Закладка* — это метка, при помощи которой можно быстро найти нужное место. Работа с закладками производится либо с панели инструментов **Edit**, либо через меню **Edit | Bookmark**.

Меню **Правка (Edit) → Закладки (Bookmarks)**



*Рис 2.6. Использование закладок*

## 2.5. Запуск кода

### Запуск программы из редактора Visual Basic

Для запуска используются кнопки и меню редактора Visual Basic:

1. В редакторе Visual Basic сделать активным процедуру (например, кликнуть мышкой в любом месте запускаемой процедуры), затем нажать F5 или в меню **Run** команду **Run Sub/UserForm**.
2. Из окна отладки по имени процедуры.

### Запуск программы из конечного приложения

Для запуска первой процедуры можно использовать следующие возможности:

- создать макрос и запустить его по имени, кнопке или комбинацией клавиш. Макрос затем может вызывать и другие процедуры;
- создать форму и воспользоваться набором событий этой формы и элементов управления на ней или просто элементом управления на листе Excel или документе Word;
- использовать событие: например, событие запуска приложения, событие открытия документа и т. п. Это рекомендованный Microsoft способ обеспечения автоматического запуска программного кода;

- можно запустить приложение из командной строки с параметром /m и именем макроса, например: winword.exe /mMyMacros

Очень удобно в этом случае использовать ярлыки, в которых можно указать этот параметр запуска.

## ***2.6. Перемещение компонентов***

Переносимые компоненты - формы (.frm), модули(.bas), модули классов(.cls).

### **Экспорт файла**

1. В Окне проекта редактора Visual Basic выбрать компонент, который необходимо скопировать.
2. В меню редактора **Файл (File) → Экспорт файла (Export File...)**.
3. Ввести имя и путь и **ОК**.

### **Импорт файла**

1. В меню редактора **Файл (File) → Импорт файла (Import File)**
2. Выбрать имя и путь сохраненного файла на диске и ОК.

## ***2.7. Практическое занятие 1: Использование макрорекодера***

На этом занятии Вы получите практические навыки по использованию макрорекодера для автоматизации простых операций и примените редактор VBA для корректировки записанного макрорекодером кода.

### **Задание 1.1 Форматирование текста в MS Word**

#### Требуется

1. Создать макрос в Microsoft Word, который автоматически форматирует выделенный текст следующим образом:
  - 1.1.Шрифт: Times New Roman, 14-й, курсивный
  - 1.2.Цвет шрифта: красный
2. Назначить вызов макроса по нажатию комбинации клавиш Alt + Ctr + Shift + A и по нажатию настраиваемой кнопки.

#### Решение

Прежде чем записывать макрос, важно очень точно спланировать свои действия. Если вы что-то сделаете не так во время записи, неправильные действия будут записаны в макрос. Например, выделять текст нужно до начала записи. Иначе в макрос попадет команда выделения текста и каждый раз после запуска программа будет выделять текст, а это по условию задачи не требуется. Или другой пример: если нужно вставить текущую дату в начало документа, может быть, имеет смысл первой командой макроса сделать переход на начало документа (<Ctrl>+<Home>).

Для записи макроса выполните следующие действия:

1. Выделите первый участок текста, который нужно отформатировать.
2. Выберите вкладку ленты **Вид** и нажмите на ней направленную вниз треугольную стрелку под надписью **Макросы** (рис. 2.7):

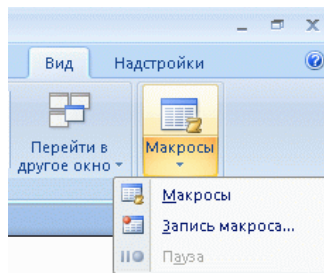


Рис. 2.7. Начало записи макроса

3. В открывшемся меню укажите команду **Запись макроса**. Появится окно для настройки свойств макроса (рис. 2.8):

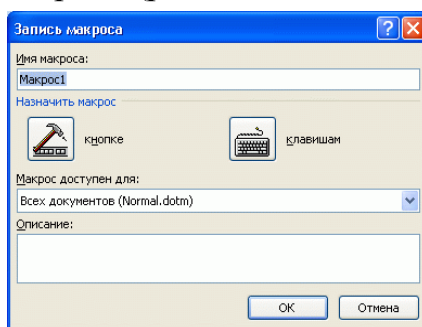


Рис. 2.8. Окно настройки свойств макроса

4. В окне диалога определите следующие параметры:

4.1. **Имя макроса**: в это поле введите имя макроса. Имена макросов должны начинаться с буквы, не должны содержать пробелов. Желательно давать макросам какие-нибудь осмысленные имена. Например, **Формат\_Times\_Красный**.

4.2. В поле **Макрос доступен для** выбирается место сохранения макроса, которое определяет его доступность для различных документов. По умолчанию указывается параметр **Всех документов (Normal.dotm)**.

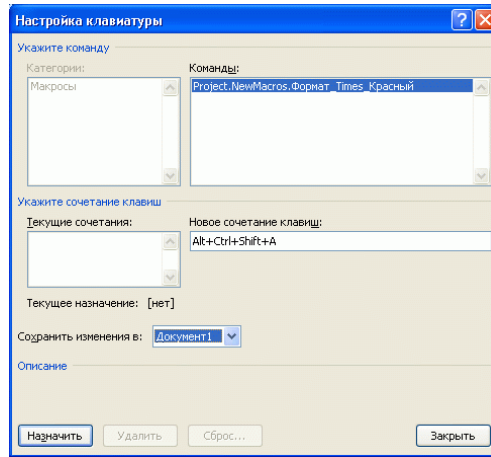
Normal.dotm — это общий шаблон, доступный для всех документов Microsoft Word. Если макрос будет сохранен в Normal.dotm — вы сможете запустить его из любого Word-документа. Следует сохранять все записываемые вами макросы в Normal.dotm лишь тогда, когда вы точно уверены в том, что макрос понадобится вам в различных документах.

Выберите в этом поле значение **Документ 1 (документ)**. Именно так называются еще не сохраненные документы. Выбрав этот пункт, макрос будет сохранен в текущем документе, то есть можно будет вызывать макрос лишь из этого документа.

4.3. В поле **Описание** содержится описание макроса, информация о том, для каких целей создается этот макрос. Оставьте его пустым.

5. Чтобы назначить макрос комбинации клавиш щелкните кнопку **клавишам**. Появится окно настройки комбинации клавиш для запуска макроса (рис. 2.9).

Для назначения макроса настраиваемой кнопке используется **Назначить макрос кнопке**. Эту операцию (как и операцию – назначение макроса клавишам) можно проделать позже.



**Рис. 2.9.** Настройка сочетания клавиши для запуска макроса

5.1. Установите курсор в поле *Новое сочетание клавиш* и нажмите нужное сочетание на клавиатуре: Alt + Ctrl + Shift + A.

В случае если это сочетание не назначено ранее для быстрого вызова каких-либо команд, под полем *Текущие сочетания* вы увидите надпись *Текущее назначение: [нет]*. Если вы увидите здесь что-нибудь другое — лучше всего поискать свободное сочетание. Иначе вы можете столкнуться с неожиданным поведением знакомых вам горячих клавиш.

5.2. В поле *Сохранить изменения в* выберите **Документ 1** — то есть наш документ, который мы в данный момент редактируем. По умолчанию здесь установлен Normal.dotm. Сохранять привязку сочетаний клавиш к макросам в Normal.dotm следует лишь в том случае, если вы, во-первых, на предыдущем шаге сохранили макрос в Normal.dotm, а во-вторых, для того чтобы данное сочетание работало во всех остальных документах MS Word, а не только в текущем документе.

5.3. Нажмите кнопку **Назначить**. Выбранное сочетание клавиш переместиться в поле *Текущие сочетания*.

5.4. Нажмите кнопку **Закреть**, завершится настройка горячих клавиш для макроса и начнется его запись. В процессе записи макроса указатель мыши дополнится значком с изображением кассеты.

6. Отформатируйте текст, пользуясь вкладками ленты Microsoft Word (щелчок правой кнопкой не действует).

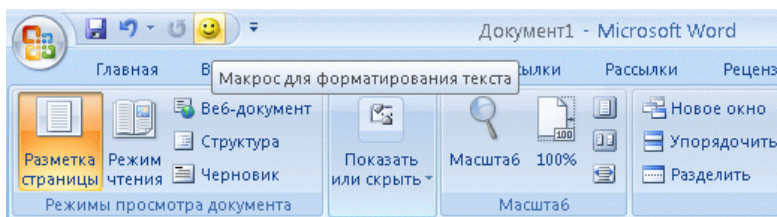
7. После настройки необходимые параметров форматирования, перейдите на вкладку **Вид** и нажмите на кнопку **Остановить запись**.

Обратите внимание на кнопку **Пауза**. С ее помощью можно приостановить запись макроса, выполнить какие-нибудь действия, которые не войдут в него, после чего возобновить запись.

После записи макроса протестируйте его работу. Для этого выделите текст и нажмите Ctrl + Alt + Shift + A. Если все сделано правильно — текст будет отформатирован.

Для настройки запуск макроса с помощью кнопки выполните следующие действия:

1. Нажмите кнопку настройки панели быстрого доступа (она находится справа от панели) и в появившемся меню выберите пункт *Другие команды*. Появится окно для настройки команд Microsoft Word.
2. В списке *Выбрать команды из* выберите **Макросы**. В поле, которое расположено ниже, появятся ссылки на доступные макросы.
3. Выделите свой ранее созданный макрос и нажмите на кнопку **Добавить>>**. Ссылка на него появится в поле **Настройка панели быстрого доступа**. В списке *Настройка панели быстрого доступа* выберите документ, который в данный момент редактируется — тем самым изменения будут внесены лишь в панель быстрого доступа этого документа.
4. Выделите строку макроса в окне *Настройка быстрого доступа* и нажмите на кнопку **Изменить**. Появится окно настройки свойств кнопки макроса.
5. Выберите понравившийся значок для кнопки макроса и введите в поле *Отображаемое имя* текст, который будет появляться при наведении на кнопку вызова макроса указателя мыши.
6. Нажмите кнопку ОК, и кнопка макроса появится на панели быстрого доступа (рис. 2.10):



**Рис. 2.10.** Кнопка для быстрого вызова макроса на панели быстрого запуска

- Сохраняя файл Microsoft Word, содержащий макросы, обязательно убедитесь в том, что вы сохраняете его в формате **.docm** (*Документ Word с поддержкой макросов*). Дело в том, что используемый по умолчанию формат **.docx** (*Документ Word*) не поддерживает макросы. Если вы сохраните документ с макросами в таком формате, результаты работы будут утеряны.

### Задание 1.2 Создание макроса в MS Excel

Процесс создания макросов в MS Excel очень похож на создание макросов в MS Word.

Представьте, что пользователь несколько раз в день передает распоряжения в различные инстанции. Каждое распоряжение должно заканчиваться, например, строками вида:

Отв. исп. Петрова М.М.		
т. 5555		

### Требуется

1. Создать при помощи макрорекордера макрос “ОтвИсп”, который бы автоматически подставлял информацию об ответственном исполнителе в активную ячейку, а информацию о телефоне — в ячейку ниже (вместо "Петрова М.М." подставьте вашу фамилию).
2. Макрос должен быть доступен для всех создаваемых документов.
3. Макрос должен запускаться по нажатию клавиш Ctrl+Shift+M.

### Решение

1. Откройте Excel и выделите на листе любую пустую ячейку.
2. Перейдите на вкладку **Вид** и в списке кнопки **Макросы** выберите пункт **Относительные ссылки**.
3. Запустите запись макроса. Откроется окно диалога **Запись макроса**.
4. В окне **Запись макроса** в поле **Имя макроса** введите имя: МояПодпись.
5. Установите указатель в поле **Сочетание клавиш** и нажмите Shift+M (значение в итоге должно выглядеть как Ctrl + Shift + M).
6. В поле **Сохранить** выберите "Эта книга" (если макрос понадобится вам не только в одном документе укажите опцию “Личная книга макросов”).
7. Нажмите на кнопку **ОК**.
8. Введите в текущую ячейку на листе Excel текст "Отв. Исп. ваши\_ФИО", например, "Отв. Исп. Петрова М.М.". Перейдите на ячейку ниже и введите текст "т. 55-55".
9. На вкладке **Вид** в списке кнопки **Макросы** выберите пункт **Остановить запись**.
10. Перейдите на другой лист и на вкладке **Вид** в списке кнопки **Макросы** выберите пункт **Макросы**. В открывшемся окне укажите требуемый макрос и нажмите на кнопку **Выполнить**. В текущую ячейку и ячейку ниже будет вставлен записанный вами текст.
11. В других ячейках проверьте работу комбинации Ctrl + Shift + M.
12. Настройте запуск макроса с помощью кнопки на панели быстрого запуска.

Сохраняя книгу Microsoft Excel, содержащую макросы, убедитесь в том, что сохраняете ее в формате "*Книга Excel с поддержкой макросов*", то есть итоговый файл имеет расширение **.xlsm**.

### **Задание 1.3** **Корректировка макроса**

#### Требуется

Изменить макрос *МояПодпись* (см. задание 1.2) так, чтобы:

- вместо инициалов выводилось только имя (полностью);
- отображался бы новый номер телефона: 373-29-59.

#### Решение

1. Откройте книгу Excel из задания 1.2.

2. Перейдите на вкладку **Вид** и в списке кнопки **Макросы** выберите пункт **Макросы**. Откроется окно **Макрос**.
3. В списке доступных макросов укажите макрос *МояПодпись* и нажмите кнопку **Изменить**. Откроется редактор VBA, который позволяет изменять макросы и создавать собственные программы.
4. Найдите в коде строку, в которой активной ячейке присваивается символьная строка “Фамилия И.О.” и измените инициалы на имя.
5. Измените номер телефона на новый: 373-29-59.
6. Закройте редактор, выбрав **File | Close and Return to Microsoft Excel**.
7. Запустите макрос и проверьте его работу.

#### **Задание 1.4 Экспорт и импорт файлов**

В этом задании вы экспортируете компонент из проекта Microsoft Word и используете его в проект Microsoft Excel.

##### Экспорт компоненты из Microsoft Word

1. Откройте файл Ex5.doc в папке \Labs\Lab02.
2. Откройте редактор Visual Basic, перейдите в окно проекта.
3. Щелкните правой кнопкой мыши на форме UserForm1.
4. Из контекстного меню выберите пункт **Экспорт файла (Export File...)**.
5. Сохраните форму на диске в файле UserForm1.frm.

##### Импорт компоненты в Microsoft Excel

1. Откройте рабочую книгу Microsoft Excel
2. Откройте редактор Visual Basic для приложения Excel, перейдите в окно проекта.
3. Если открыто несколько книг, выберите тот проект, куда Вы хотите импортировать форму.
4. В меню редактора Visual Basic, выберите **Файл (File) → Импорт файла (Import File)**. Найдите сохраненную в предыдущем задании форму и импортируйте ее.
5. Протестируйте форму. Для этого щелкните по кнопке **Запуск подпрограммы/UserForm (Run Sub/UserForm)**



## 3. Использование Visual Basic for Application

### 3.1. Основы синтаксиса

Синтаксис VBA почти полностью совпадает с синтаксисом Visual Basic. Основные синтаксические принципы этого языка следующие:

- VBA нечувствителен к регистру;
- чтобы закомментировать код до конца строки, используется одинарная кавычка (') или команда REM;
- символьные значения должны заключаться в двойные кавычки ("");
- максимальная длина любого имени в VBA (переменные, константы, процедуры) — 255 символов;
- начало нового оператора — перевод на новую строку (точка с запятой, как в C, Java, JavaScript, для этого не используется);
- ограничений на максимальную длину строки нет (хотя в редакторе в строке помещается только 308 символов). Несколько операторов в одной строке разделяются двоеточиями:  
MsgBox "Проверка 1" : MsgBox "Проверка 2"
- для удобства чтения можно объединить несколько физических строк в одну логическую при помощи пробела и знака подчеркивания после него:

```
MsgBox "Сообщение пользователю" _
```

```
& vUserName
```

### 3.2. Переменные, типы данных, константы

#### Использование переменных. Типы переменных

Переменная - именованная область памяти, где могут храниться различные данные, которые можно изменять во время выполнения программы.

Переменная имеет следующие характеристики:

- имя переменной. Используя имя, можно обращаться к переменной в программе.
- тип данных, которые могут храниться в переменной. Тип определяет характер данных, которые мы можем хранить в переменной. Например, это могут быть числовые данные (возраст пользователя) и строковые данные (имя пользователя).

Имена переменных должны:

- Начинаться с буквы.
- Состоять из букв, цифр и символа подчеркивания "\_".
- Быть уникальными в пределах области видимости.
- Не должно совпадать с ключевыми словами Visual Basic.
- Быть не длиннее 255 символов.

*Примечание.* Ключевыми словами называют слова, являющиеся элементами языка Visual Basic. В число ключевых слов входят имена

инструкций (например, **If** и **Loop**), функций (например, **Len** и **Abs**) и операторов (например, **Or** и **Mod**).

Определение типа данных задает:

- область возможных значений типа;
- структуру организации данных;
- операции, определенные над данными этого типа.

Таблица 3.1: Типы переменных. Основные характеристики

Тип данных	Размер	Диапазон значений	Префикс
Byte (байт)	1 байт	От 0 до 255.	byt
Boolean (логический)	2 байт	True или False.	bln
Integer (целое)	2 байт	От -32 768 до 32 767.	int
Long (длинное целое)	4 байт	От -2 147 483 648 до 2 147 483 647.	lng
Single	4 байт	От -3,4E38 до -1,4E-45 для отриц.знач.; от 1,4E-45 до 3,4E38 для пол.знач.	sng
Double	8 байт	От -1,7E308 до -4,9E-324 для отриц.знач.; от 4,9E-324 до 1,7E308 для пол.знач.	dbl
Currency (денежный)	8 байт	С фиксир.точкой От -922 337 203 685 477,5808 до 922 337 203 685 477,5807.	cur
Date (даты и время)	8 байт	От 1 января 100 г. до 31 декабря 9999 г.	dtm
Object (объект)	4 байт	Ссылка на объект (указатель)	obj
String (строка перем.длины)	10 байт + длина строки	От 0 до приблизительно 2 миллиардов.	str
String (строка пост.длины)	Длина строки	От 1 до приблизительно 65 400.	str
Variant	не менее 16 байт	Любой из перечисл.выше объектов, Null,Error,Empty,Nothing	var

Примечание. На основе встроенных типов программист может строить собственные, им определенные типы данных.

#### Объявление переменных

При объявлении переменной определяется ее тип и область видимости.

- Когда переменные не объявлены явно, они имеют тип Variant.
- Предпочтительнее использовать явное объявление переменных.
- Для объявления переменных используются операторы **Dim**, **Public**, **Private** и **Static**.

- При объявлении переменных желательно использование префиксов и заглавных букв.

Пример.

**Dim** intTemp As Integer

*Замечание.* Всегда используйте явное описание переменных.

Преимущества объявления переменных:

- сокращается количество ошибок: программа с самого начала откажется принимать в переменную значение неправильного типа (например, строковое вместо числового);
- при работе с объектами подсказка по свойствам и методам действует только тогда, когда мы изначально объявили объектную переменную с нужным типом.

VBA не требует явно объявлять переменные. Но если добавить оператор **Option Explicit** в общую область модуля, то все переменные должны быть объявлены явно.

Оператор **Option Explicit** можно добавлять автоматически в каждый новый модуль:

**Сервис (Tools) → Параметры (Options) → вкладка Редактор (Editor) → флажок Явное описание переменных (Require Variable Declaration)**

Всегда помещайте объявление переменных в начало блока.

#### Инициализация переменных

VBA инициализирует переменные в момент их объявления:

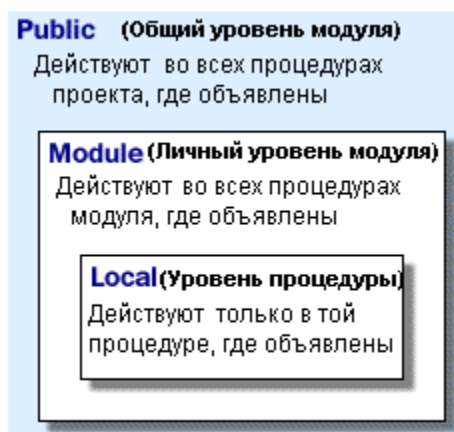
- 0 – для числовых значений;
- пустая строка «» - для строк переменной длины;
- строка, содержащая нули – для строк фиксированной длины;
- Empty – для переменных типа Variant;

#### Область видимости

В VBA предусмотрены следующие ключевые слова для определения области видимости переменных (см. рис.3.1):

- Dim — Если переменная объявлена как Dim в области объявлений модуля, то она будет доступна во всем модуле, если в процедуре — только на время работы этой процедуры (используется в большинстве случаев).
- Private — при объявлении переменных в стандартных модулях VBA значит то же, что и Dim. Отличия проявляются только при создании своих классов.
- Public — такая переменная будет доступна всем процедурам во всех модулях данного проекта, если вы объявили ее в области объявлений модуля. Если вы объявили ее внутри процедуры, она будет вести себя как Dim.
- Static — такие переменные можно использовать только внутри процедуры. Эти переменные видны только внутри процедуры, в которой они объявлены, зато они сохраняют свое значение между разными вызовами

этой процедуры. Обычно используются для накопления каких-либо значений.



*Рис 3.1. Область видимости (определения) переменных, процедур и объектов.*

**Local.** Объявлены ключевым словом **Dim** в теле процедуры.

```
Sub MyProcedure()  
    Dim strTempName As String  
    strTempName = "John Doe"  
    MsgBox "Welcome Back, " & strTempName & "!"  
End Sub
```

**Module.** Переменные, объявленные ключевым словом **Private (Dim)** в Общей области раздела Описаний модуля или формы, доступны только в процедурах этого модуля.

```
Private strTest As String
```

**Public.** Переменные, объявленные ключевым словом **Public** в Общей области раздела Описаний модуля.

```
Public intCounter As Integer
```

### Время жизни переменной

Время жизни переменной определяет, как долго переменная хранится в памяти.

Для изменения времени жизни локальной переменной используется ключевое слово **Static**, что делает локальную переменную статической.

Пример.

```
Sub Count()  
    Static intCount As Integer  
    intCount = intCount + 1  
    MsgBox "The count is now " & intCount  
End Sub
```

### Константы

Для каждого пользовательского типа есть соответствующие ему константы.

Синтаксис оператора **Const**, используемого для объявления именованных констант:

```
[Public | Private] Const <имя константы> [As type] = <константное _ выражение>
```

Пример.

```
Public Const pi As double = 3.141593
```

Константы бывают двух видов:

- **ПОЛЬЗОВАТЕЛЬСКИЕ КОНСТАНТЫ;**  
 Const PI = 3.14159                    ‘ объявление константы  
 Area = PI \* radius ^ 2                ‘ использование константы  
 Circum = 2 \* PI \* radius
- **встроенные константы** (связанные с тем или иным приложением, обычно имеют префикс wd, xl, ac...)

Например, встроенная константа VbAbortRetryIgnore (прервать – повторить – отменить) применяется в функции MsgBox.

**Пример. Использование констант VBA**

```
Sub MsgBoxTest()
    Dim intResult As Integer
    intResult = MsgBox("Хотите продолжить?",vbYesNoCancel + _
        vbExclamation, "Приложение MyApp")
    Select Case intResult
        Case vbYes
            MsgBox "Пользователь выбрал ДА"
        Case vbNo
            MsgBox "Пользователь выбрал НЕТ"
        Case vbCancel
            MsgBox " Пользователь выбрал ОТМЕНА"
    End Select
End Sub
```

**3.3. Процедуры**

**Использование процедур Sub и Function**

В VBA можно выполнить только тот программный код, который содержится в какой-либо процедуре (обычной в стандартном модуле, событийной для элемента управления на форме и т. п.).

В VBA предусмотрены следующие **типы процедур**:

- процедура типа **Sub (подпрограмма)** — универсальная процедура для выполнения каких-либо действий:  
 Макрос в VBA — это процедура типа Sub, не имеющая параметров. Только макросы можно вызывать по имени из редактора VBA или из приложения Office. Все другие процедуры нужно вызывать либо из других процедур, либо специальными способами, о которых будет рассказано далее;
- процедура типа **Function (функция)** — набор команд, которые должны быть выполнены. Принципиальное отличие: функция возвращает вызвавшей ее программе (или процедуре) какое-то значение, которое будет там использовано.

Таблица 3.2: Создание и вызов процедуры Sub и Function

Пример <b>Sub</b> процедуры.	Вызов созданной <b>Sub</b> процедуры.
<pre>Sub GetData()     ' [... Ваш код ...] End Sub</pre>	<pre>Sub BeginProcess()     GetData End Sub</pre>

Пример создания <b>Function</b> процедуры	Вызов созданной <b>Function</b> процедуры
<pre>Function OpenFile () As Boolean ' открытие файла If ' [открытие успешно]     OpenFile = True Else     OpenFile = False End If End Function</pre>	<pre>blnSuccess = OpenFile() If blnSuccess = True Then     MsgBox "Успешно" Else     MsgBox "Файл не открыт" End If</pre>

Таблица 3.3: Синтаксис инструкции Sub и Function

Синтаксис инструкции <b>Sub</b>	Синтаксис инструкции <b>Function</b>
<pre>[Private   Public] [Static] <b>Sub</b> имя [(списокАргументов)]     [инструкции]     [Exit Sub]     [инструкции] End Sub</pre>	<pre>[Private   Public] [Static] <b>Function</b> имя [(список_Аргументов)] [As тип]     [инструкции]     [имя = выражение]     [Exit Function]     [инструкции]     [имя = выражение] End Function</pre>

Синтаксис инструкций Sub и Function содержит следующие элементы:

Элемент	Описание
Public	Указывает, что процедура Sub (Function) доступна для всех других процедур во всех модулях. При использовании в личном модуле (модуле, который содержит инструкцию Option Private) такая процедура является недоступной вне проекта.
Private	Указывает, что процедура Sub (Function) доступна для других процедур только того модуля, в котором она описана.
Static	Указывает, что локальные переменные процедуры Sub (Function) сохраняются в промежутках времени между вызовами этой процедуры.
имя	Имя процедуры Sub (Function), удовлетворяющее стандартным правилам именования переменных.
список_Аргументов	Список переменных, представляющий аргументы, которые передаются в процедуру Sub (Function) при ее вызове. Имена переменных разделяются запятой.
инструкции	Любая группа инструкций, выполняемых в процедуре Sub (Function).
тип	Тип данных значения, возвращаемого процедурой Function, поддерживаются все типы (за исключением строк фиксированной длины).
выражение	Возвращаемое значение процедуры Function.

Параметр *Список\_Аргументов* имеет следующий синтаксис и элементы:

[Optional] [ByVal | ByRef] [ParamArray] *имяПеременной*[( )] [As тип] [= поУмолчанию]

Элемент	Описание
Optional	Указывает, что аргумент не является обязательным. При использовании этого элемента все последующие аргументы, которые содержит список_Аргументов, также должны быть описаны с помощью ключевого слова <b>Optional</b> . Не допускается использование ключевого слова <b>Optional</b> для любого из аргументов, если используется ключевое слово <b>ParamArray</b> .
ByVal	Указывает, что этот аргумент передается по значению.
ByRef	Указывает, что этот аргумент передается по ссылке (по умолчанию).
ParamArray	Используется только в качестве последнего элемента в списке список_Аргументов, позволяет задавать произвольное количество аргументов. Оно не может быть использовано со словами <b>ByVal</b> , <b>ByRef</b> или <b>Optional</b> .
Имя Переменной	Имя переменной, удовлетворяющее правилам именования переменных.
тип	Тип данных аргумента, переданного в процедуру; поддерживаются все типы (за исключением строк фиксированной длины).
поУмолчанию	Любая константа или выражение, дающее константу. Используется только вместе с параметром <b>Optional</b>

### Передача аргументов

Пример описания и вызова процедуры с использованием аргументов:

```
Function FtoC(Temperature As Single) As Single
    ' Перевод температуры из шкалы по Фаренгейту в шкалу по Цельсию
    FtoC = (Temperature - 32) * (5 / 9)
End Function
```

Вызов **FtoC** function:

- Из окна отладки (Immediate): ?FtoC(100)

- Из процедуры:

```
Private Sub ConvertFtoC()
    MsgBox "98 Degrees F is " & FtoC(98) & " C"
End Sub
```

Пример описания процедуры с необязательным аргументом:

```
Sub DisplayContactInfo(strName As String, Optional strNickname)
    ' в зависимости от того, передан аргумент или нет, выводятся разные
    ' сообщения
    If IsMissing(strNickname) Then
        MsgBox strName & " goes by no other name..."
    Else
        MsgBox "You can refer to " & strName & " as " & strNickname
    End If
End Sub
```

Пример описания процедуры с неизвестным числом аргументов

```
Sub DisplayList(ParamArray parMyList() As Variant)
```

```

For intCount = LBound(parMyList) To UBound(parMyList)
    MsgBox parMyList(intCount)
Next intCount
End Sub

```

#### Пример вызова Sub или Function с именованными аргументами

```

UpdateSales CustID:=45, ProductID:=22 (вызов Sub или Function)
Call UpdateSale (CustID:=22, ProductID:=4) (вызов Sub)

```

#### Пример вызова функции

```

blnSuccess = OpenFile ("myfile.dat") ' с последующим использованием ее значения
OpenFile "myfile.dat" ' без использования возвращаемого значения

```

#### **Основные функции языка**

- Проверки типов (TypeName, IsDate, IsEmpty, IsError, IsNull, IsNumeric, IsObject...)
- Преобразования типов (CBool, CInt, CLng, CDb1,..., Fix, Int)
- Преобразования данных (Chr, Asc, DateValue...)
- Математические функции (Abs, Mod, Cos, Rnd...)
- Функции обработки строк (StrComp, Len, Instr, Mid, Left, Trim...)
- Функции обработки дат и времени (DateSerial, Day, Month, Year)

### **3.4. Управляющие инструкции**

**Инструкция** - синтаксически завершенная конструкция, представляющая отдельное действие, описание или определение. Обычно, инструкция занимает отдельную строку программы, хотя допускается использование двоеточия (:) для размещения в одной строке нескольких инструкций.

Допускается также использование пробела и символа продолжения строки ( ) для продолжения одной логической программной строки на нескольких физических строках.

#### **Условный оператор If...Then...Else**

Задаёт выполнение определенных групп инструкций в зависимости от значения выражения. Синтаксис:

```

If условие Then [инструкции] [Else инструкции_else]

```

Или

```

If условие Then
[инструкции]
[ElseIf условие-n Then
[инструкции_elseif] ...
[Else
[инструкции_else]]
End If

```

**Пример.**

```

If strUserPassword = "Triangle" Then
' [Вход пользователя в систему]
Else
' [Сообщение о неверном пароле]
End If

```



## Оператор выбора Select Case

Выполняет одну из нескольких групп инструкций в зависимости от значения выражения.

Инструкция **Select Case** может служить альтернативой инструкции **ElseIf** в **If...Then...Else** при оценке одного выражения, которое имеет несколько возможных значений. В то время как **If...Then...Else** для каждой инструкции **ElseIf** оценивает разные выражения, инструкция **Select Case** оценивает выражение только один раз, в начале управляющей структуры.

Синтаксис:

```
Select Case выражение
[Case списокВыражений-n
  [инструкции-n]] ...
[Case Else
  [инструкции_else]]
End Select
```

Пример.

```
Sub InputExpense()
  Dim strUserName As String
  strUserName = InputBox("Введите свое имя")
  Select Case strUserName
    Case "Teresa"
      MsgBox "Доступ автора"
    Case "James"
      MsgBox "Доступ пользователя"
    Case Else
      MsgBox "В доступе отказано", vbCritical
  End Select
End Sub
```

**Цикл** – действия процедуры, повторяющиеся заданное количество раз или пока выполняется или не выполняется некоторое условие.

Процесс выполнения все операторов, заключенных в структуру цикла, один раз называется **итерацией** цикла.

Структуры цикла, всегда выполняющиеся заданное количество раз, называются циклами с **фиксированным** числом итераций.

Другие типы структур цикла повторяются переменное количество раз в зависимости от некоторого набора условий. Такие циклы называются **неопределенными** циклами.

Блок операторов, находящийся между началом и концом цикла называется "**тело цикла**".

Таблица 3.4: Виды циклов

Название цикла	Вид
For - Next	С фиксированным количеством повторов. Выполняется заданное количество раз
Do While - Loop While - Wend	С предусловием. Если не верно условие, заданное на входе в цикл, может не выполниться ни разу
Do - Loop	С постусловием. Выполняется по меньшей мере один раз.

## Цикл Do...Loop

Повторяет выполнение набора инструкций, пока условие имеет значение True или пока оно не примет значение True.

Синтаксис:

```
Do [{While | Until} условие]
[инструкции]
[Exit Do]
[инструкции]
Loop
```

Пример.

```
Do Until EndOfFile = True
' [Читать следующую строку в файле]
' [Установить EndOfFile=True, если достигнут конец файла]
Loop
```

## Do While - Loop

```
Do
[инструкции]
[Exit Do]
[инструкции]
Loop [{While | Until} условие]
```

Пример.

```
Sub GatherCriteria()
Dim blnCheck As Boolean
Dim strMyCriteria As String, strTempString As String
strMyCriteria = "MyCriteria = " ' инициализация критерия
Do
strTempString = InputBox("Введите критерий")
If UCase(strTempString) = "DONE" Then
blnCheck = True
strMyCriteria = Left(strMyCriteria, (Len(strMyCriteria) - 2))
Else ' пользователь не закончил ввод
blnCheck = False
strMyCriteria = strMyCriteria & strTempString & ", "
End If
Loop Until blnCheck = True
MsgBox strMyCriteria
End Sub
```

## Цикл For...Next

Повторяет выполнение группы инструкций указанное число раз.

Синтаксис:

```
For счетчик = начало To конец [Step шаг]
[инструкции]
[Exit For]
[инструкции]
Next [счетчик]
```

Пример.

```
Sub ComputeAverage()
Dim intNumberOfStudents As Integer, intCount As Integer
Dim sngScore As Single, sngTotalScore As Single
```

```

Dim sngAverage As Single

intNumberOfStudents = InputBox("Сколько студентов?")
For intCount = 1 To intNumberOfStudents
    sngScore = CSng(InputBox("Введите балл"))
    sngTotalScore = sngTotalScore + sngScore
Next intCount
sngAverage = sngTotalScore / intNumberOfStudents
MsgBox "Средний балл: " & Str(sngAverage)
End Sub

```

### Цикл While...Wend

Повторяет выполнение последовательности операторов, пока заданное условие не станет ложным.

Синтаксис:

```

While условие
[инструкции]
Wend

```

### Цикл For Each...Next

Повторяет заданную последовательность операторов для каждого элемента массива или набора (коллекции).

Синтаксис:

```

For Each элемент In группа
[инструкции]
[Exit For]
[инструкции]
Next [элемент]

```

## 3.5. Обработка ошибок

### Типы ошибок

Можно выделить два типа ошибок, с которыми сталкивается программист:

- ошибки, которые сопровождают создание программ,
- ошибки времени выполнения.

#### Ошибки, возникающие при создании программ

- **Синтаксические** – если неправильно будет введен оператор или ключевое слово, если не указать часть выражения. Многие синтаксические ошибки "отлавливаются" редактором кода VBA еще в процессе ввода кода. Об обнаружении других ошибок сообщается в ходе компиляции и запуска программы. При этом компилятор VBA выдает информацию о том, в какой строке кода обнаружена ошибка и в чем она заключается.
- **Логические** – связаны с неправильными формулами расчета показателей, неверным использованием переменных и т.д. В ходе выполнения программа ведет себя не так, как планировалось. Главное здесь — найти

причину неправильного поведения программы. Обычно для выявления и исправления ошибок такого типа предназначены приемы отладки.

#### Ошибки при выполнении программы

Такие ошибки называют еще ошибками времени выполнения.

Они возникают, когда в процессе выполнения программа столкнулась с проблемой, решить которую она не в состоянии.

Происходят они, как правило, при неправильном вводе данных пользователем, при возникновении обстоятельств, делающих дальнейшую нормальную работу программы невозможной. Например, ошибку вызовет попытка использовать текстовые данные в арифметическом выражении, попытка сохранения файла в несуществующей директории, деление на ноль и т.д.

Ошибки времени выполнения возникают в нормально работающих программах, которые прошли проверку на синтаксическую и логическую правильность.

С этими ошибками можно бороться, используя один из двух методов.

1. Разработка программы таким образом, чтобы не допустить этих ошибок, создание программных конструкций, которые предотвращают возникновение ошибок.
2. Перехват ошибок и их обработка.

При создании обработчика ошибок, Вы используете три составляющие:

- включение обработчика ошибок (перехват ошибки – установка ловушки);
- написание обработчика;
- возврат из обработчика.

#### **Включение обработчика ошибок**

Есть три формы инструкции On Error:

- Включение - **On Error GoTo Label**
- Отключение - **On Error GoTo 0**
- Передача управления следующему оператору **On Error Resume Next**

Синтаксис:

Включение (On Error GoTo Label) и отключение (On Error GoTo 0) обработчика ошибок:

```
Sub SomeHandling()  
    On Error GoTo Catch_Errors  
    '[Часть процедуры, где м.б. ошибка]  
    On Error GoTo 0  
    '[Другая часть процедуры]  
    Exit Sub  
Catch_Errors:  
    '[Обработчик ошибок]  
End Sub
```

## Общие принципы обработки ошибок:

1. Перед опасным кодом (сохранение или открытие файла, возможность деления на ноль и т. п.) помещается команда:

```
OnErrorGoToметка_обработчика_ошибки
```

например:

```
Dim a As Integer, b As Integer, c As Integer
On Error GoTo ErrorHandlerDivision
c = a / b
```

2. Далее в коде программы помещается метка обработчика ошибки и программный код обработки:

```
ErrorHandlerDivision:
MsgBox "Ошибкаприделении"
```

3. Поскольку в такой ситуации код обработчика ошибки будет выполняться даже в том случае, если ошибки не было, есть смысл поставить перед меткой обработчика команду **Exit Sub** (если это процедура) или **Exit Function**(если это функция).

Полный код программы может выглядеть так:

```
Private Sub UserForm_Click()
    Dim a As Integer, b As Integer, c As Integer
    On Error GoTo ErrorHandlerDivision
    c = a / b
    Exit Sub
ErrorHandlerDivision:
    MsgBox "Ошибкаприделении"
End Sub
```

Как правило, если есть возможность исправить ошибку, то в обработчике ошибок ее исправляют (или предоставляют такую возможность пользователю), если нет — то выдают пользователю сообщение с объяснением и прекращают работу программы.

4. После выполнения кода обработчика ошибки вам нужно будет сделать выбор: либо продолжить выполнение той процедуры, в которой возникла ошибка, либо прекратить ее выполнение и передать управление вызвавшей ее процедуре. В вашем распоряжении три варианта:
  1. еще раз выполнить оператор, вызвавший ошибку (если обработчик ошибки может определить характер возникшей проблемы). Для этого в обработчик ошибок вставить команду **Resume**;
  2. пропустить оператор, вызвавший ошибку. Для этой цели можно использовать команду **Resume Next**;
  3. продолжить выполнение с определенного места в программе. Для этого используется команда **Resume метка**. Синтаксис работы с меткой — такой же, как в операторе GoTo.

Отметим еще несколько моментов, которые связаны с обработкой ошибок:

- чтобы вернуться в нормальный режим работы после прохождения опасного участка кода (отменить обработку ошибок), можно воспользоваться командой:

```
On Error GoTo 0
```

- в вашем распоряжении имеется также команда **On Error Resume Next**. Она предписывает компилятору просто игнорировать все возникающие ошибки и переходить к выполнению следующего оператора. На практике очень часто перед выполнением опасного оператора используется эта команда, а затем при помощи конструкции **Select Case** проверяется номер возникшей ошибки (через свойства объекта **Err**), и в зависимости от этого организуется дальнейшее выполнение программы.

Пример. Обработка ошибки в теле процедуры без обработчика ошибок и передачи управления следующему оператору (**On Error Resume Next**)

```
Sub TrapErrorsInLine()
' ошибка обрабатывается в процедуре
On Error Resume Next
Dim intNumberOfTests As Integer, intTotalPoints As Integer
Dim lngAverage As Long
'
intTotalPoints = InputBox("Введите общее число баллов")
intNumberOfTests = InputBox("Введите число экзаменов")
' подсчет среднего
lngAverage = intTotalPoints / intNumberOfTests
' проверка деления на ноль
If intNumberOfTests <= 0 Then
MsgBox "Средний балл сосчитан быть не может."
Else
MsgBox "Средний балл = " + Str(lngAverage)
End If
End Sub
```

### Объект Err

Глобальный объект **Err** содержит информацию об ошибках выполнения.

#### Свойства объекта Err

<b>Number</b>	Возвращает или задает числовое значение, представляющее код ошибки. Свойство <b>Number</b> является свойством объекта <b>Err</b> , используемым по умолчанию (уникально).
<b>Description</b>	возвращает или задает текст сообщения, соответствующего определенной ошибке.
<b>Source</b>	возвращает или задает имя объекта или приложения, в котором возникла ошибка.
<b>HelpFile</b>	Возвращает или задает строковое выражение, определяющее полный путь к файлу справочной системы.
<b>HelpContext</b>	Возвращает или задает строковое выражение, содержащее контекстный идентификатор раздела в файле справочной системы.
<b>LastDLLError</b>	Возвращает системный код ошибки, возникшей при вызове библиотеки динамической компоновки (DLL). Доступно только для чтения.

## Методы объекта **Err**

<b>Clear</b>	Очищает все значения свойств объекта Err.
<b>Raise</b>	Создает ошибку выполнения.

### Пример. Использование свойств и методов объекта Err

```
Sub ErrObject()  
    On Error Resume Next  
    '  
    Err.Raise InputBox("Введите номер ошибки")  
    MsgBox "Ошибка: " & Str(Err.Number) & _  
        vbCrLf & "описание = " & Err.Description  
    ' очистка объекта Err  
    Err.Clear  
End Sub
```

В обработчиках ошибок обычно используют инструкцию **Select Case** (или подобные) для идентификации ошибки и выполнения различных действий в зависимости от этого.

Для того, чтобы исключить обработку ошибок в тех случаях, когда ошибки не возникают, в теле процедуры перед меткой подпрограммы обработки ошибок необходимо поместить соответственно одну из следующих инструкций: **Exit Function**, **Exit Sub** или **Exit Property**

### Пример обработчика ошибок:

```
Sub ErrorTrap()  
    On Error GoTo ErrorHandler  
    Dim intNumberOfTests As Integer  
    Dim intTotalPoints As Integer  
    Dim lngAverage As Long  
    intTotalPoints = InputBox("Введите общее число баллов ")  
    intNumberOfTests = InputBox("Введите число экзаменов")  
    lngAverage = intTotalPoints / intNumberOfTests  
    MsgBox "Средний балл: " & Str(lngAverage)  
    Exit Sub  
ErrorHandler:  
    Select Case Err.Number  
        Case 11      ' проверка деления на 0  
            MsgBox " Деление на 0 невозможно "  
            intNumberOfTests = InputBox("Введите число экзаменов")  
            Resume  
        Case 13      ' ошибка преобразования типов  
            MsgBox " Должны быть введены числовые значения"  
        Case Else    ' any other error  
            MsgBox Err.Description & " произошла"  
    End Select  
End Sub
```

### 3.6. Отладка (Debugging)

#### Средства отладки

#### Режим прерывания (останова)

**Останов** – пауза в исполнении кода. Во время останова программа не заканчивает свою работу, а переходит в режим ожидания, т.е. программа Visual Basic остается загруженной, но после выполнения одной инструкции следующая инструкция не выполняется.

Программу в режим паузы можно перевести следующими способами:

- запустить программу в режиме пошагового выполнения (меню **Debug | Step Into** или клавиша <F8>). В этом случае программа будет переходить в режим паузы после выполнения каждого оператора;
- установить в программе точку останова (breakpoint).
- нажав во время работы программы комбинацию клавиш **Ctrl + Break**.
- воспользоваться контролируемым выражением (в окне **Watches**).

Когда программа остановлена, вы можете выполнить следующие действия:

- Просмотреть значения переменных, наведя на них указатель мыши.
- Продолжить выполнение программы в режиме **Step Into (Шаг с заходом)** - выбрав соответствующую команду меню или нажав клавишу **F8**.
- Отредактировать программу.
- Продолжить исполнение программы в обычном режиме командой **Run** о **Sub/User Form (Запустить | Процедуру/Форму)**, нажатием клавиши **F5** или соответствующей кнопкой на панели инструментов
- Остановить выполнение программы командой **Run** о **Reset (Запустить о Перезагрузка)** или кнопкой на панели инструментов
- Воспользоваться другими средствами отладки - окнами **Immediate**, **Locals**, **Watch**.

Помимо режима **Step Into** существуют следующие режимы отладки, доступные в меню **Debug**:

- **Step Over (Шаг с обходом)** (Перейти на следующую строку). Эта команда полезна при отладке программы, содержащей вызовы уже отлаженных процедур.

В режиме **Step Over** отладчик не входит в процедуру, выполняя ее без отладки, после чего переходит на следующую строку.

- **Step Out (Шаг с выходом)** (Выполнить процедуру) - эта команда позволяет выполнить текущую процедуру (например, вызванную из кода основной программы при обычной отладке) без остановки в каждой строке. Следующая остановка будет сделана на строке, которая следует за вызовом процедуры в основном тексте программы.
- **Run To Cursor (Выполнить до курсора)** - выполняет программу до позиции, на которой установлен курсор. Аналогично установке одиночной точки останова.



## Окна отладки

Окно **Immediate** (Немедленное выполнение) предназначено для немедленного выполнения программного кода.

Чтобы отобразить его, можно воспользоваться командой **View | Immediate Window (Вид | Окно непосредственных изменений)**.

С помощью окна **Immediate** можно:

- просматривать и изменять значения переменных и свойств объекта,
- мгновенно выполнять различные команды,
- вызывать процедуры и функции - как пользовательские, так и встроенные,
- организовать вывод различных данных в ходе выполнения программы.

Два назначения окна отладки:

1) Выполнение отдельных операций во время останова в окне отладки

```
? intCount      ' печать значения переменной
? X + Y         ' печать значения выражения
? CStr(FtoC(32)) ' печать значения, возвращаемого функцией
Proc           ' вызов процедуры или функции
```

2) Вывод результатов отладочной печати из программы в окно отладки (**Immediate**)

```
Debug.Print "Salary = " & curSalary
```

### Другие окна

- Окно локальных переменных (**Locals Window**)  
В этом окне выводятся значения всех переменных и свойств объектов, доступных в настоящий момент
- Окно контрольного значения (**Watch Window**)  
Окно предназначено для контроля за отдельными выражениями, значениями переменных и т.д.

## 3.7. Практическое занятие 2: Использование Visual Basic for Applications

На этом занятии вы напишете код Visual Basic for Applications, обработаете ошибки времени выполнения и изучите средства отладки редактора Visual Basic (Visual Basic Editor).

### Задание 2.1 Разработка процедуры с помощью макрорекодера

#### Требуется

Реализовать в Microsoft Word процедуру, которая запрашивала фамилию исполнителя и выводила полученную информацию на экран. Основу процедуры создать с помощью макроса.

#### Решение

1. Начните запись макроса, назовите его **ЗапросИсполнителя**, укажите, что макрос доступен только для текущего документа, нажмите ОК.
2. Введите слово *Исполнитель* и остановите запись.

3. Откройте окно **Макросы** нажатием соответствующей кнопки на вкладке **Вид**, выберите макрос **ЗапросИсполнителя** и нажмите на кнопку **Изменить**. Откроется окно редактора Visual Basic с открытым в нем макросом:

```
Sub ЗапросИсполнителя()  
,  
' ЗапросИсполнителя Макрос  
,  
,  
    Selection.TypeText Text:="Исполнитель"  
End Sub
```

4. Добавьте над первой строкой Selection.TypeText следующий код:

```
Dim sInput As String  
sInput = InputBox("Введите фамилию и инициалы исполнителя", "Запрос данных")
```

5. Измените строку Selection.TypeText Text:="Исполнитель" на следующую:

```
Selection.TypeText Text:=( "Исполнитель " & sInput)
```

6. Сохраните измененный макрос, закройте окно редактора кода и проверьте работу макроса.

### **Задание 2.2 Создание процедуры типа Function**

#### Требуется

Реализовать функцию для получения значения стоимости продукта, которая вычисляется как сумма цены и налога. Налог равен произведению цены и налоговой ставки.

#### Решение

##### ➤ Создание процедуры типа функция

1. Откройте редактор Visual Basic.
2. Вставьте новый модуль.
3. Создайте функцию с именем **TotalCost** с одним аргументом **dblPrice** типа **Double**, которая возвращает значение типа **Double**.
4. В теле функции опишите константу **dblTaxRate**, содержащую налоговую ставку, равную 0.085.
5. Вычислите общую стоимость, складывая цену продукта **dblPrice** и налог, полученный умножением цены на налоговую ставку.
6. Вычислите общую стоимость в качестве возвращаемого значения функции.

Возможный вариант решения:

```
Function TotalCost(dblPrice As Double) As Double  
' 8.5% - налоговая ставка  
Const dblTaxRate = 0.085  
TotalCost = dblPrice + (dblPrice * dblTaxRate)  
End Function
```

##### ➤ Тестирование функции

1. Создайте процедуру типа **Sub**.
2. Вызовите функцию **TotalCost** и передайте ей значение цены, равное 10.

Выведите результат в Окно отладки (Immediate Window).

Для того чтобы увидеть результат, сделайте **Окно отладки (Immediate Window)** видимым **Вид (View) → Окно отладки (Immediate Window)** (или CTRL +G)

Возможный вариант решения:

```
Sub GetTotal()  
Dim dblTotal As Double  
    dblTotal = TotalCost(10)  
Debug.Print dblTotal  
End Sub
```

В окне отладки (**Immediate Window**) протестируйте эту функцию с различными значениями (например, ?TotalCost(100))

### Задание 2.3 Создание процедуры типа Sub

Требуется: Написать процедуру подсчета и анализа стоимости в зависимости от вводимой цены.

Решение: Создание процедуры типа Sub:

1. В этом же модуле, который использован в задании, создайте процедуру типа **Sub** с именем **DisplayTotalCost**.
2. Опишите две переменных типа **Double** с именами **dblPrice** – цена и **dblTotalCost** - стоимость.
3. Присвойте переменной **dblPrice** значение, возвращаемое функцией **InputBox**.
4. Присвойте переменной **dblTotalCost** значение, возвращаемое функцией **TotalCost** из задания 1 для вычисления стоимости, в зависимости от значения переменной **dblPrice**.
5. Используйте условный оператор для проверки, является ли значение стоимости меньшим, чем 50. Используя функцию **MsgBox**, выведите стоимость и сообщение о допустимости цены.

Возможный вариант решения:

```
Sub DisplayTotalCost()  
' Exercise 2  
Dim dblTotalCost As Double  
Dim dblPrice As Double  
    dblPrice = InputBox(«Пожалуйста, введите цену»)  
    dblTotalCost = TotalCost(dblPrice)  
If dblTotalCost < 50 Then  
    MsgBox «Стоимость » & Str(dblTotalCost) & vbCrLf & _  
        «Это меньше допустимой», vbCritical  
Else  
    MsgBox «Стоимость » & Str(dblTotalCost) & vbCrLf & _  
        «Такая стоимость допускается», vbExclamation  
End If  
End Sub
```

6. Протестируйте процедуру

### Задание 2.4 Использование циклических и условных структур

Требуется: Написать функцию, которая использует циклическую структуру для получения набора значений от пользователя.

Решение: Создание новой функции:

1. Напишите новую функцию с именем **GetSum**, которая возвращает целое значение.
2. Опишите две переменных с именами **intSum** и **intUserData**.
3. Задайте начальное значение **intSum** равное нулю.
4. Используйте функцию **InputBox** для ввода пользователем числа и присвоение результирующего значения переменной **intUserData**.
5. Добавьте новое значение к сумме **intSum**.
6. Создайте циклическую структуру, повторяя шаги 4 и 5, до тех пор, пока сумма не станет больше 25.
7. Функция **GetSum** должна возвращать значение полученной суммы.

Возможный вариант решения:

```
Function GetSum() As Integer
' Exercise 3 - get integers until sum > 25
Dim intUserData As Integer
Dim intSum As Integer
    intSum = 0
    Do While intSum <= 25
intUserData = InputBox(«Пожалуйста, введите целое число», _
        «Ввод пользователя», 0)
        intSum = intSum + intUserData
    Loop
GetSum = intSum
End Function
```

➤ Тестирование функции

Вариант 1:

1. Создайте новую процедуру **Sub**.
2. Вызовите функцию **GetSum** и напечатайте возвращаемое значение в окне **Immediate**.

```
Sub Test()
    i = GetSum
        Debug.Print i
End Sub
```

Вариант 2:

В окне отладки (Immediate Window)

```
?GetSum
```

### **Задание 2.5 Обработка ошибок времени выполнения**

Требуется

Добавить процедуру обработки ошибок в функцию **GetSum**.

1. Активизируйте обработчик ошибок в начале функции **GetSum** для перехвата ошибок времени выполнения. Добавьте обработку ошибок к вашему коду.
2. Если ошибка произойдет, управление должно перейти к обработчику ошибок, в котором выводится окно сообщения, объясняющее ошибку пользователю. Предусмотрите возврат из обработчика ошибок и продолжение выполнения процедуры в случае ошибки № 13. Если

произойдет другая ошибка, выполнение функции необходимо остановить.

### Решение

```
Function GetSum() As Integer
    Dim intUserData As Integer
    Dim intSum As Integer
    On Error GoTo GetSum_ErrorHandler
    intSum = 0
    Do While intSum <= 25
        intUserData = InputBox(«Пожалуйста, введите целое число», _
        «Ввод пользователя», 0)
        intSum = intSum + intUserData
    Loop
    GetSum = intSum
Exit Function
GetSum_ErrorHandler:
Select Case Err.Number
Case 13
' Ошибка преобразования (type mismatch)
MsgBox «Это значение прибавлено быть не может.» & _
«Введите целое число», vbExclamation
intUserData = 0
Resume Next
Case Else
MsgBox «Произошла ошибка: » & Err.Description _
& vbCrLf & «...Работа функции заканчивается», vbCritical
Exit Function
End Select
End Function
```

### Тестирование обработчика ошибок

1. Запустите функцию. Когда появится окно запроса о вводе целого значения, введите строку и посмотрите, что произойдет.  
Если вы ввели строку, появится ошибка, связанная с преобразованием типов. Вы не можете назначить строку целочисленной переменной. Вы можете использовать функцию **IsNumeric** VBA для определения, содержит ли переменная числовое значение.
2. Когда вас попросят ввести целочисленное значение, введите **75000**, и посмотрите, что произойдет.  
Если вы ввели **75000**, то произойдет ошибка переполнения. Целочисленная переменная не может иметь значение 75000.

### Задание 2.6 Использование средств отладки

В этом задании вы познакомитесь со средствами отладки Visual Basic.

- Работа с панелью инструментов Отладка (Debug) и окном Контрольного значения (Watch Window)
  1. Выберите пункт меню **Вид (View)** → **Панель управления (ToolBars)** → **Отладка (Debug)**.

2. В панели выберите кнопку Окно контрольного значения (Watch Window).

➤ Пошаговое выполнение кода

1. Установите точку останова в начале функции **GetSum** из предыдущего задания.
2. Запустите функцию.
3. Для пошагового выполнения операторов **While...loop**, щелкните по кнопке **Шаг с заходом (Step Into)** в панели **Отладка (Debug)** или нажмите F8.
4. Переместите индикатор текущей выполняемой строки к последней строке кода этой функции.

Текущая строка выполняемого кода отмечается желтым цветом и боковым индикатором в форме желтой стрелки.

➤ Вычисление переменных на этапе выполнения

1. Установите точку прерывания в начале функции **GetSum** из предыдущего задания.
2. Запустите функцию.
3. В состоянии прерывания (Break mode), выделите переменную **intUserData** и затем щелкните **Контрольное значение (Quick Watch)** в панели **Отладка (Debug)**.
4. Для добавления переменной к окну **Контрольного значения (Watch Window)** щелкните **Добавить (Add)**.
5. Нажмите F8 для пошагового выполнения кода и наблюдайте за изменением значения **intUserData** в окне **Контрольного значения (Watch Window)**.
6. Выделите переменную **intSum**, и затем щелкните **Добавить контрольное значение (Add Watch)** в контекстном меню.
7. Установите тип контрольного значения равный **Останов при изменении значения выражения (Break When Value Changes)**.
8. Удалите точку останова и запустите функцию заново. Процедура должна прерваться, когда значение **intSum** изменится.

➤ Изменение значений на этапе выполнения

1. Установите точку прерывания в начале функции **GetSum** из предыдущего задания.
2. Запустите функцию.
3. В состоянии прерывания задайте значение для переменной в окне **Отладка (Immediate Window)**.

В окне **Отладки (Immediate Window)** вы можете назначить значение переменной: `intSum = 30`

4. В Окне контрольные значения (**Watch Window**) посмотрите, что изменилось.
5. Измените значение **intSum** в окне контрольные значения.
6. Посмотрите это значение в окне **Отладка**: `?intSum`

## 4. Формы и объекты управления

### 4.1. Введение: элементы управления в формах и документах

Применение форм в приложении VBA происходит следующим образом:

1. В большинстве случаев форма запускается при открытии пользователем документа.
2. Далее пользователь выполняет на форме какие-то действия по вводу или выбору информации (например, выбирает значения в раскрывающемся списке, устанавливает значения для флажков и переключателей и т. п.).
3. Затем, как правило, нажимает на кнопку на этой форме, и введенная им информация передается в базу данных, отправляется по электронной почте, записывается в файл для распечатки и т. д.

Многие элементы управления можно вставлять непосредственно на страницу документа (рис. 4.1), однако на практике чаще всего применяется вариант, при котором элементы управления размещаются на форме (рис. 4.2). Вне зависимости от того, используется ли форма или элементы управления размещаются напрямую в документе, набор элементов управления и приемы работы с ними одинаковы.

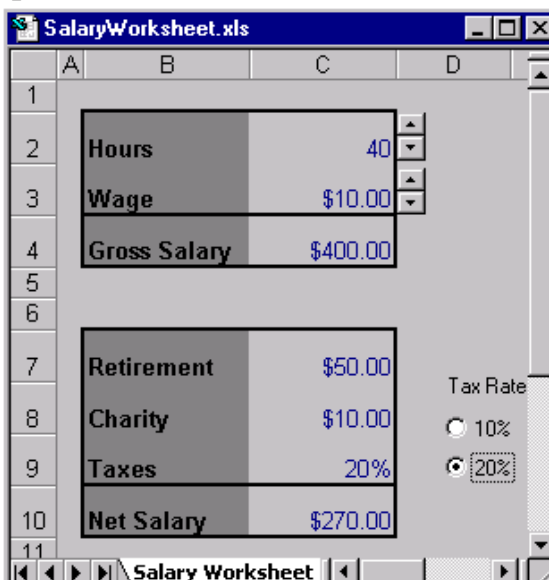


Рис 4.1 Размещение элементов управления в документе

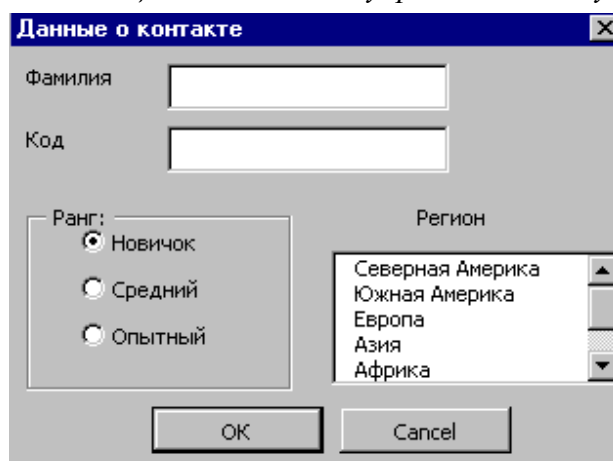


Рис 4.2 Форма. Размещение элементов управления в форме.

## 4.2. Использование элементов управления в формах

### Добавление формы (диалогового окна) в проект

Новые пользовательские диалоговые окна проектируются в редакторе VBA. В процессе создания диалогового окна вначале определяется его видимое изображение на экране – форма с управляющими элементами. Затем задаются их свойства и разрабатываются процедуры, обрабатывающие события, связанные с формой и ее элементами управления.

Для добавления формы указать имя проекта в окне проекта, в который добавляется форма и в меню редактора Visual Basic выбрать:

#### **Insert (Вставка) → UserForm;**

В проект можно добавить произвольное число форм.

Для просмотра и редактирования свойств формы:

- выделить форму
- в меню выбрать Вид (View) → Окно свойств (Properties Window) или (F4)

Наиболее часто используемые **свойства формы** (кроме **ShowModal** все они применимы и для других элементов управления):

**Name** — определяет имя формы, используемое только программистом в коде для этой формы (и в окнах редактора Visual Basic). После создания формы ее имя, предлагаемое по умолчанию (UserForm1), рекомендуется заменить на что-нибудь более значимое, чтобы было проще ориентироваться в программе (это относится также ко всем элементам управления).

**Caption** — определяет заголовок формы (по умолчанию совпадает с именем формы). Рекомендуется ввести строку, которая будет напоминать пользователю о назначении формы (например, "Выбор типа отчета").

**Enabled** — если это свойство установлено в False, пользователь не сможет работать с формой. Используется для временного отключения формы, например, пока пользователь не обеспечит какие-то условия для ее работы.

**ShowModal** — если свойство установлено в True (по умолчанию), то пользователь не может перейти к другим формам или вернуться в документ, пока не закроет эту форму (модальный режим работы).

Большая часть других свойств относится к внешнему виду, размерам и местонахождению формы.

### Добавление элементов управления в форму

**Элемент управления** — это специализированный объект, который можно размещать на формах VBA (или непосредственно в документах) и который используется для организации взаимодействия с пользователем.

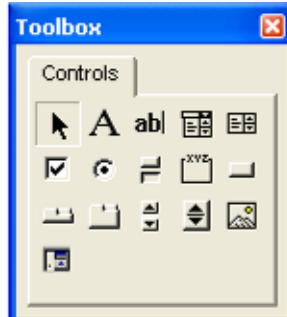
Для того чтобы открыть панель элементов, надо выбрать пункт меню редактора Visual Basic **View (Вид) → Toolbox (Панель инструментов)**

Для добавления на форму элементов управления надо выбрать элемент на панели инструментов **Toolbox** (см. рис.4.3),а затем:

- Либо кликнуть мышкой в форме;
- Либо "нарисовать" мышкой;



- Уточнить положение элементов в окне можно командами меню **Format (Формат)**.



*Рис 4.3. Панель элементов управления (Toolbox).*

#### **Задание порядка перехода по TAB**

Задать порядок обхода управляющих элементов в форме можно одним из двух способов:

- В редакторе Visual Basic: **View (Вид) → Tab Order (Последовательность перехода)**.
- Использовать свойство **TabIndex** элемента управления.

Если свойству элемента управления **TabStop** присвоить значение **False**, то этот элемент никогда не будет активным.

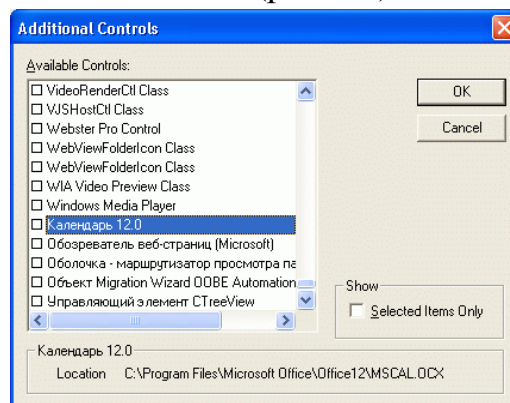
#### **Добавление дополнительных элементов управления на панель элементов управления**

На панели инструментов **Toolbox** в редакторе VBA отображаются не все элементы управления.

Просмотреть установленные в системе элементы и вывести кнопки для их вызова на панель инструментов можно одним из следующих способов:

- щелкните правой кнопкой мыши по панели и в появившемся меню выберите пункт **Additional Control (Дополнительные элементы управления)**.
- меню **Tools (Сервис) → Additional Controls**

Откроется окно **Additional Control** (рис.4.4).



*Рис. 4.4 Список доступных элементов управления (Available Controls)*

#### **Установка свойств элементов управления**

- на этапе проектирования,
- во время выполнения программы (например, `TextBox1.Text = "Hello"`)

### Добавление кода к событиям элементов управления

Для перехода в окно редактора VBA с целью ввода текста процедуры обработки события, щелкните дважды объект, к которому это событие относится (элемент управления или форму). Появится окно «Code». В раскрывающемся списке слева вверху перечислены все объекты формы. При выборе объекта в этом списке справа вверху появляются события, связанные с данным объектом. Например, на рис. 4.5. показан список событий для командной кнопки.

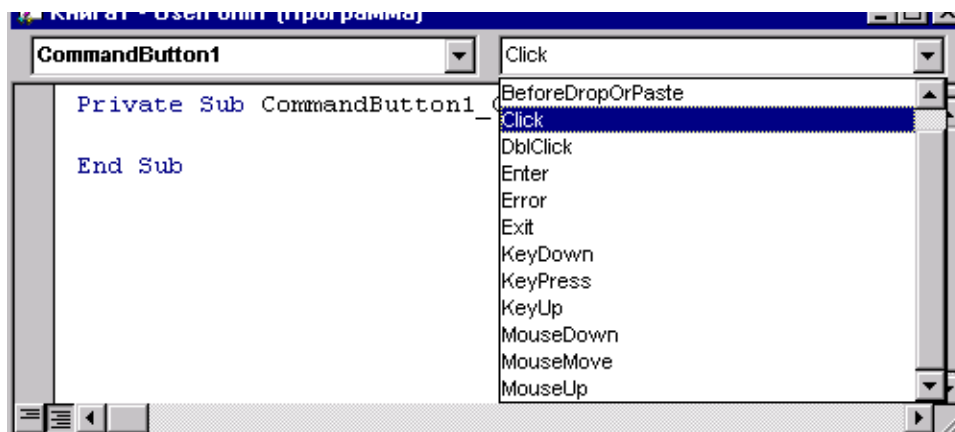


Рис 4.5. Окно программы для командной кнопки.

События, для обработки которых процедуры уже написаны, выделены жирным шрифтом. Для создания новой процедуры или редактирования существующей, надо щелкнуть в списке соответствующее событие. Все элементы управления имеют процедуры обработки событий. Имена этих процедур образуются из имени элемента управления, знака подчеркивания ( ) и названия события. Например, **CommandButton1\_Click**.

### 4.3. Работа с формой

Диалоговые окна Office 2010 могут работать в режиме как модального, так и немодального диалога. Формы обладают важным свойством **ShowModal**, позволяющим установить модальность диалогового окна.

#### Вызов формы

Метод Show

```
[UserForm.]Show modal
```

modal – аргумент, определяющий модальность, может принимать значения **vbModal** (по умолчанию) или **vbModeless**

Если форма уже была загружена в память, она просто станет видимой, если нет — то будет автоматически загружена (произойдет событие **Load**).

Пример.

```
Private Sub GetUserName ()  
    UserForm1.Show  
End Sub
```

#### Закрытие формы

После того, как пользователь введет, или выберет нужные данные на форме и нажмет требуемую кнопку, форму необходимо убрать. Для этого можно воспользоваться двумя способами:

1. Спрятать форму (использовать метод **Hide()**), например:

```
UserForm1.Hide
```

Форма будет убрана с экрана, но останется в памяти. Потом при помощи метода **Show()** можно будет опять ее вызвать в том же состоянии, в каком она была на момент "прятанья", а можно, например, пока она спрятана, программно изменять ее и расположенные на ней элементы управления. Окончательно форма удалится из памяти при закрытии документа.

2. Если форма больше точно не потребуется, можно ее удалить из памяти при помощи команды **Unload**:

```
Private Sub cmdOK_Click()  
    ActiveDocument.Content.InsertAfter txtUserName.Text  
    Unload UserForm1  
End Sub
```

### Работа с событиями формы

Событие формы	Когда происходит
Initialize	Когда форма загружается в память
Activate	Когда форма получает фокус
DeActivate	Когда форма теряет фокус
QueryClose	Когда пользователь закрывает форму. Вы можете отменить закрытие формы
Terminate	Когда форма закрывается (выгружается)

### Инициализация элементов управления

Начальные значения элементов управления могут задаваться:

- на этапе конструирования,
- используя программный код на этапе выполнения программы:
  - обработка события Initialize
  - в процедуре перед выводом формы

Обработка события Initialize:

```
Private Sub GetUserName ()  
    UserForm1.Show  
End Sub  
Private Sub UserForm_Initialize()  
    UserForm1.lstNames.AddItem "Test One"  
    UserForm1.lstNames.AddItem "Test Two"  
End Sub
```

Непосредственно в процедуре перед выводом формы:

```
Private Sub GetUserName ()  
    UserForm1.txtSalesPersonID.Text = strCurrentID  
    UserForm1.Show  
....  
End Sub
```

### Сохранение данных формы

При окончании работы с формой, данные, вводимые в форме, больше не хранятся в памяти. Можно использовать кнопки ОК - для сохранения данных, введенных в форме, и Cancel - для отмены сохранения.

Пример. Завершение работы с формой.

Код в модуле:

```
Public strRegion As String
Public intSalesPersonID As Integer
Public blnCancelled As Boolean
Sub LaunchSalesPersonForm()
frmSalesPeople.lstRegions.AddItem "North"
frmSalesPeople.lstRegions.AddItem "South"
frmSalesPeople.Show
If blnCancelled = True Then
    MsgBox "Информация не сохраняется", vbExclamation
Else
    MsgBox "Код пользователя: " & Str(intSalesPersonID) & _
        " Регион: " & strRegion
End If
End Sub
```

Код в форме:

```
' – пользователь нажал Cancel
Private Sub cmdCancel_Click()
    Module1.blnCancelled = True
    Unload Me
End Sub
' – пользователь нажал ОК
Private Sub cmdOK_Click()
    'Сохранение данных
    Module1.intSalesPersonID = txtSalesPersonID.Text
    Module1.strRegion = lstRegions.List(lstRegions.ListIndex)
    Module1.blnCancelled = False
    Unload Me
End Sub
' – Инициализация формы
Private Sub UserForm_Initialize()
    Module1.blnCancelled = True
End Sub
```

### Контроль данных

При разработке формы рекомендуется реализовать контроль вводимых пользователем данных.

Для этого можно использовать:

- события элементов управления.

Пример.

```
Private Sub txtSalesPersonID_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    If Not IsNumeric(txtSalesPersonID.Text) Then
        MsgBox "Пожалуйста, введите код пользователя – целое число"
        Cancel = True
    End If
End Sub
```

- процедуры для контроля данных.

Пример.

```
Private Sub cmdOK_Click()
    If Not IsNumeric(txtSalesPersonID.Text) Then
```

```

MsgBox " Пожалуйста, введите код пользователя – целое число "
txtSalesPersonID.SetFocus
Exit Sub
End If
If lstRegions.ListIndex = -1 Then
MsgBox "Необходимо выбрать регион из списка"
lstRegions.SetFocus
Exit Sub
End If
'Сохранение данных
intSalesPersonID = txtSalesPersonID.Text
strRegion = lstRegions.List(lstRegions.ListIndex)
Module1.blnCancelled = False
Unload Me
End Sub

```

#### 4.4. Использование элементов управления в документах

Панель элементов управления располагается на ленте основного приложения: вкладка **Разработчик** → Группа **Элементы управления**.

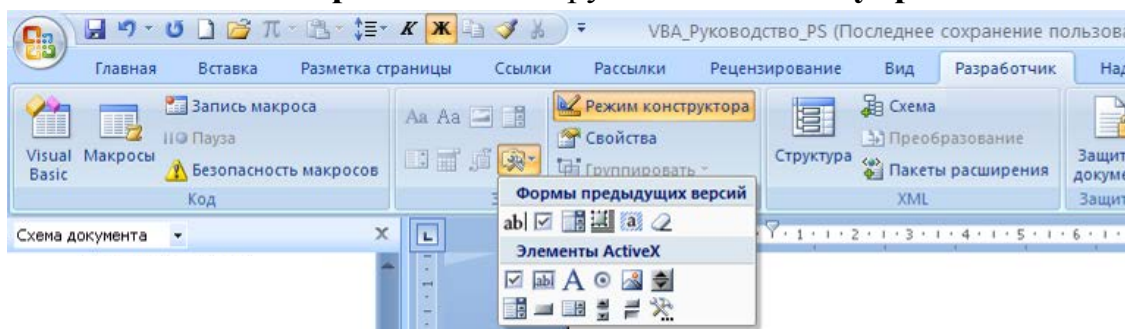


Рис 4.6. Панель элементов управления на вкладке ленты **Разработчик**.

Для добавления элементов управления в документ следует выбрать мышью элемент управления на панели и кликнуть или "нарисовать" в документе. Переход между режимами осуществляется нажатием соответствующей кнопки **Режим конструктора**. Для установки свойств элементов управления следует выбрать соответствующую команду:

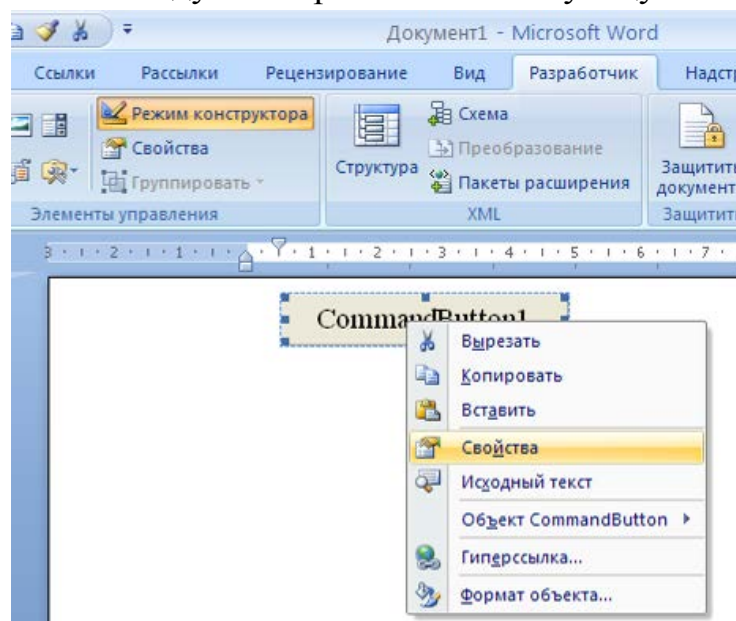


Рис 4.7. Открытие **ОД Свойства**.

#### 4.5. Практическое занятие 3: Использование элементов управления

На этом занятии вы будете работать с элементами управления в формах и документах.

##### Задание 3.1 Заставка при загрузке книги Excel

###### Требуется

Реализовать возможность вывода на экран заставки при открытии любой заданной книги в Excel.

###### Решение

1. Создание экранной формы.

- Откройте редактор Visual Basic и создайте новую форму, используя команду меню **Insert - Form**. Появится пустая серая оконная форма будущей заставки. Добавьте на нее изображение при помощи панели инструментов **Toolbox** (меню **View - Toolbox**):

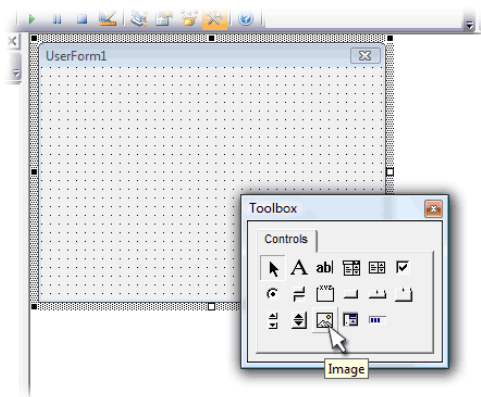


Рис 4.8 Пустая форма.

- Нажмите кнопку *Image* и растяните на форме прямоугольник - в него будет помещено фоновое изображение. Затем на панели инструментов **Properties** (меню **View - Properties**) задайте выберите файл картинки в поле *Picture*:

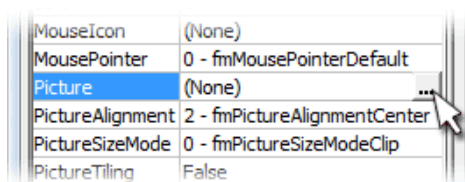


Рис 4.9 Выбор файла картинки в ОД Свойства.

Возможно, придется немного изменить размер формы, чтобы изображение уместилось полностью.

Чтобы написать на форме текст, можно использовать элемент управления *Label* с панели **Toolbox**.

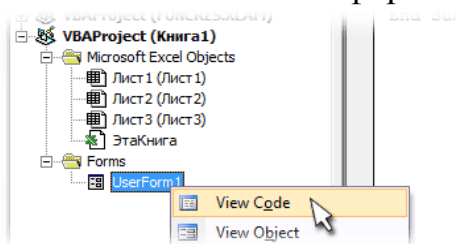
Ну, и наконец, выделив предварительно всю форму, можно задать текст в строке заголовка, используя свойство *Caption* в панели **Properties**.

2. Добавление управляющего кода

- Вставьте событийную процедуру для объекта *Workbook* (*ЭтаКнига*), на процедурном листе в окне выбора объектов (вверху слева) выберите объект *Workbook* и добавьте в него следующий код:

```
Private Sub Workbook_Open()
    UserForm1.Show
End Sub
```

- Затем щелкните правой кнопкой мыши по форме и выберите **View Code**:



**Рис 4.10** Открытие модуля кода для формы.

- В открывшийся модуль формы добавьте следующий код:

```
Private Sub UserForm_Activate()
    Application.OnTime Now + TimeValue("00:00:05"), "KillTheForm"
End Sub
```

- И, наконец, вставьте обычный модуль (**Insert - Module**) и добавьте код:

```
Private Sub KillTheForm()
    Unload UserForm1
End Sub
```

- Закройте редактор Visual Basic, сохраните файл и откройте книгу.

При открытии книги Excel выполняет процедуру *Workbook\_Open* из модуля *ЭтаКнига*. Эта процедура отображает на экране нашу форму-заставку. При отображении формы запускается процедура *UserForm\_Activate*, которая с задержкой в 5 секунд запускает макрос *KillTheForm*, который убирает форму с экрана.

### **Задание 3.2 Убегающая кнопка**

#### Требуется

1. Создать форму шириной 350 пикселей и высотой 200 пикселей.
2. Разместить на форме кнопки "Да", "Нет" и надпись: "Хотите ли вы получить прибавку к зарплате?".
3. Настроить свойства кнопки с надписью "Да" таким образом, чтобы на нее нельзя было перевести фокус ввода.
4. Написать программу, которая при наведении указателя мыши на кнопку с надписью "Да" перемещала бы эту кнопку в пределах формы.

#### Решение

1. Создайте новую форму, установите следующим образом ее свойства:

Name (Имя) — frm\_MovingButton  
 Width (Ширина) — 350  
 Height (Высота) — 200  
 Caption (Заголовок формы) — Тест

2. Добавьте на форму следующие элементы управления:

Название	Name (Имя)	Caption (Надпись)	Tabstop (Фокус ввода по нажатию клавиши Tab)	Height (Высота)	Width (Ширина)
Кнопка №1	cmd_Yes	Да	False	30	80
Кнопка №2	cmd_No	Нет		30	80
Надпись	lbl_text	Хотите ли вы получить прибавку к зарплате?			

3. Дважды щелкните по cmd\_Yes. В редакторе кода выберите в поле событий MouseMove.

4. В тело обработчика события MouseMove для кнопки cmd\_Yes введите:

```
cmd_Yes.Left = Rnd * 250
```

```
cmd_Yes.Top = Rnd * 150
```

В этом коде мы обращаемся к следующим свойствам кнопки cmd\_Yes:

- Left — расстояние между кнопкой и левым краем формы.
- Top — расстояние между кнопкой и верхним краем формы.

Функция Rnd возвращает случайное число от 0 до 1. Для вычисления нового расстояния от кнопки до левого края формы умножаем случайное число на 250, до верхнего края формы – на 150. Эти значения взяты не случайно. Ниже приведен их расчет.

$250 = (\text{ширина формы}) - (\text{ширина кнопки}) - (\text{запас по ширине}) = 350 - 80 - 20$

$150 = (\text{высота формы}) - (\text{высота кнопки}) - (\text{запас по высоте}) = 200 - 30 - 20$

Каждый раз, когда пользователь наводит указатель мыши на кнопку cmd\_Yes, кнопка случайным образом меняет положение на форме. Значит, с помощью мыши он не сможет нажать на кнопку.

Кнопки на формах можно нажимать не только с помощью мыши. Если на кнопке установлен фокус ввода (как вы знаете, он перемещается по элементам управления по нажатию клавиши Tab на клавиатуре), "нажать" на кнопку можно, нажав клавишу Пробел или Enter. Свойство Tabstop кнопки cmd\_Yes мы установили равным False — то есть кнопка не сможет получить фокус ввода и ее нельзя будет нажать даже с использованием клавиатуры.

Для того чтобы программа приобрела законченный вид, добавим обработчики события Click для кнопок cmd\_No и cmd\_Yes.

Нажатие на кнопку cmd\_No должно вывести надпись: "Спасибо за участие в опросе" и завершить программу. А если пользователю все же удастся нажать на неуловимую cmd\_Yes (это возможно, если кнопка "прыгнет" так, что окажется под указателем мыши), должна выводиться какая-нибудь подходящая надпись.

Для удобства работы с программой добавьте на лист MS Excel кнопку cmd\_Show\_Form, которая вызывала бы форму программы.



Используйте метод Show формы frm\_MovingButton, чтобы отобразить ее по нажатию на кнопку:

```
Frm_MovingButton.Show
```

### Задание 3.3 Добавление элементов управления к форме

В этом задании вы создадите новую форму для ввода информации о покупателе.

#### ➤ Создание новой формы

1. В Microsoft Excel создайте новую рабочую книгу и затем откройте редактор Visual Basic (**Alt + F11**).
2. Добавьте новую форму к проекту (**Вставка (Insert)→UserForm**). Переименуйте ее (в свойстве **Name** для формы дайте имя, например Contact), измените заголовок формы (свойство **Caption**, например, «Данные о контакте»).
3. Добавьте элементы управления к форме (смотри рис. 4.11).

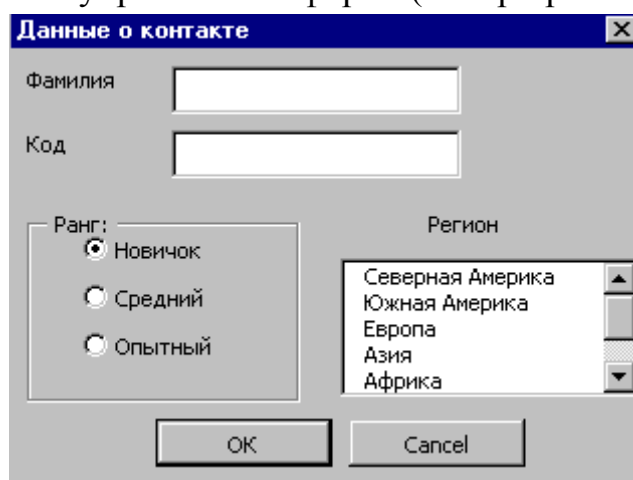


Рис 4.11 Общий вид формы.

- Для этого необходимо с панели элементов (если она не видна выберите в меню **Вид (View) → Панель элементов(Toolbox)**) перетащить необходимые объекты управления: два поля (TextBox), три переключателя (OptionButton), рамку (Frame), объединяющую переключатели, список (ListBox), три надписи (Label), две кнопки (CommandButton).
4. Назовите объекты управления, отформатируйте их, чтобы они выглядели красиво и установите порядок обхода элементов. Для установки порядка обхода элементов можете воспользоваться свойством элемента управления **TabIndex** (начинается с 0) для каждого элемента, а можете использовать форму **Последовательность перехода (Tab Order)** (при выделенной форме в меню **Вид (View) → Последовательность перехода (Tab Order)**)
  5. Добавьте код к событию **Initialize** формы, так чтобы в нем добавлялись значения к списку и по умолчанию был выделен переключатель (OptionButton) - **Новичок**.

```
Private Sub UserForm_Initialize()  
' св-во Name списка lstRegion  
' св-во Name первого переключателя OptB1
```

```

        ' код покупателя – TxtId
        ' фамилия покупателя - TxtName
lstRegion.AddItem "Северная Америка"
lstRegion.AddItem "Южная Америка"
lstRegion.AddItem "Европа"
lstRegion.AddItem "Азия"
lstRegion.AddItem "Африка"
lstRegion.AddItem "Австралия"
lstRegion.AddItem "Антарктика"
    OptB1 = True
End Sub

```

6. Добавьте код к событию **Click** для кнопки **ОК**, в котором выводится окно сообщения, содержащее данные, введенные в форме.

```

Private Sub CmdOK_Click()
    Dim strData As String
        strData = "Введена следующая информация" & vbCrLf & vbCrLf
        strData = strData & " Фамилия: " & TxtName.Text & vbCrLf
        strData = strData & " Код: " & TxtId.Text & vbCrLf
strData = strData & " Регион: " & lstRegion.List(lstRegion.ListIndex) & vbCrLf
    If OptB1 Then
        strData = strData & " Опыта работы нет" & vbCrLf
    ElseIf OptB2 Then
        strData = strData & " Небольшой опыт работы" & vbCrLf
    ElseIf OptB3 Then
        strData = strData & " Специалист" & vbCrLf
    End If
    ' вывод сообщения
    MsgBox strData, vbOKOnly, "Информация о контакте"
End Sub

```

7. Добавьте код к событию **Click** кнопки **Cancel** для закрытия формы.

```

Private Sub CmdCancel_Click()
    Unload Me
End Sub

```

- Протестируйте новую форму

1. Запустите новую форму.
2. Введите любые данные в форму и щелкните ОК.

- Используйте события документа для загрузки формы

1. В событии **Workbook\_Open** добавьте код для вывода формы.

```

Private Sub Workbook_Open()
    Contact.Show
End Sub

```

2. Закройте и откройте рабочую книгу.

Форма должна открываться при открытии книги.

### Задание 3.4 Контроль данных

В этом задании вы добавите программные коды для проверки вводимых данных.

- Контроль объектов управления

1. Откройте форму, созданную в предыдущем задании.

2. Добавьте код к событию **Exit** для поля ввода кода покупателя (в примере txtID), проверяющий является ли идентификационный номер числовым. Если код покупателя - не число, должно выводиться сообщение и пользователь не должен покинуть поле, пока не введет корректную информацию.

```
Private Sub TxtId_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    If Not IsNumeric(TxtId.Text) Then
        MsgBox "Пожалуйста, введите числовой код"
        Cancel = True
    End If
End Sub
```

3. Запустите форму, введите код, содержащий не числовые символы, и попытайтесь покинуть поле. Вы должны получить сообщение об ошибке.

➤ Контроль данных на уровне формы

1. Добавьте код к событию **Click** кнопки **OK** для проверки, выбрал ли пользователь значение из списка (после оператора Dim).
2. Если свойство **ListIndex** для списка равно -1 (т.е. пользователь не выбрал регион), предложите пользователю выбрать регион из списка.
3. Добавьте код, проверяющий, введена ли фамилия покупателя.

```
' проверка, выбран ли регион
If lstRegion.ListIndex = -1 Then
    MsgBox "Пожалуйста, выберите регион"
    lstRegion.SetFocus
End Sub

If Len(txtName.Text) = 0 Then
    MsgBox "Пожалуйста, введите фамилию"
    txtName.SetFocus
End Sub
```

4. Запустите форму, затем щелкните ОК.  
Вы получите сообщение о необходимости выбора региона.
5. Выберите регион и щелкните ОК.  
Вы получите сообщение о том, что не введена фамилия покупателя.

### Задание 3.5 Добавление элемента управления в документ

В этом задании вы добавите объекты управления в документ, позволяющие пользователю изменить формат документа.

➤ Добавление объектов управления к документу Microsoft Excel

1. Откройте Microsoft Excel и создайте новую книгу.
2. Используя панель управления (в меню Microsoft Excel выберите **Вид** → **Панели инструментов** → **Панель управления**) разместите два переключателя в документе.
3. В окне свойств каждого переключателя (из контекстного меню переключателей - пункт свойства) задайте **Name** (например, optStyleOne и optStyleTwo).

4. Задайте свойство **Caption** переключателей соответственно StyleOne и StyleTwo.

➤ Добавление данных к рабочему листу

Добавьте данные, представленные в таблице, в ячейки рабочего листа.

	Восток	Запад
Зима	80	90
Весна	70	88
Осень	40	75

➤ Создайте макрос изменяющую формат

1. Выберите пункт меню **Сервис** → **Макрос** → **Начать запись**.
2. Назовите первый макрос Style1.
3. В макросе выполните следующее:
  - a. Отметьте данные, добавленные к рабочему листу.
  - b. В пункте меню **Формат** выберите **Автоформат**.
  - c. В окне диалога **Автоформат** выберите формат и щелкните ОК.
  - d. Отметьте любую ячейку для снятия отметки с данных.
4. В плавающей панели щелкните кнопку **Остановить запись**
5. Повторите эти шаги для записи другого макроса с именем Style2, выбрав другой формат.

➤ Добавление кода к переключателям

1. В событии **Click** для переключателя StyleOne вызовите макрокоманду Style1.
  - a. Сделайте правый щелчок мыши по переключателю и выберите пункт **Исходный текст**.
  - b. Введите **Style1** в процедуре обработки события **Click**.

```
Private Sub optStyleOne_Click()
```

```
    Style1
```

```
End Sub
```

2. В процедуру обработки события **Click** переключателя StyleTwo вызовите макрокоманду Style2.

➤ Протестируйте работу переключателей

1. Щелкните кнопку **Выход из конструктора** для выхода из режима конструктора.
2. Щелкните по переключателю. Формат должен поменяться выбранным образом.

## 5. Объектные модели и автоматизация

### 5.1. Введение. Объекты, классы, объектные модели

#### Объект

Данные и код, манипулирующий этими данными, объединены в структуре, называемой **классом**.

**Класс** является обобщением понятия *тип данных* и задает свойства и поведение **объектов** класса – экземпляров класса. Каждый **объект** уникален, **класс** описывает все объекты определенного типа. Каждый объект принадлежит определенному классу. Отношение между объектом и его классом такое же, как между переменной и ее типом.

Класс обладает свойствами и методами.

**Свойства** описывают характеристики объекта.

**Методы** – операции преобразования свойств.

Наряду со свойствами и методами, с классом связывается еще одно понятие – **событие**.

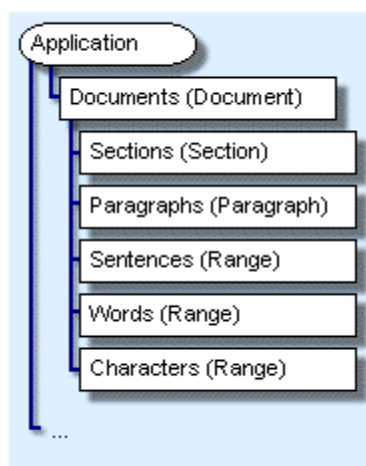
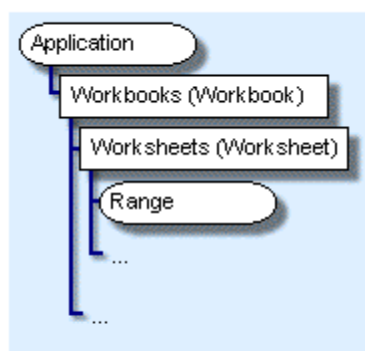
Каждый класс имеет определенный набор событий, которые могут возникать при работе с объектами класса, как при определенных действиях пользователя, так и как результат действия системы.

**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.

#### Объектная модель

В VBA определены специальные объектные модели для каждого **приложения** Microsoft Office и объектные модели, общие для всех компонентов Microsoft Office 2010. Все приложения обладают иерархической моделью (т.е. какие объекты входят в приложение и их связь между собой).

Объектная модель Microsoft Excel    Объектная модель Microsoft Word



Любая объектная модель, определенная в Visual Basic for Applications, состоит из объектов, каждый из которых представляет собой отдельный объект или коллекцию (семейство) объектов.

#### Модули классов

Кроме большого числа заранее определенных классов, Visual Basic for Applications позволяет программисту создавать собственные классы.

Синтаксически класс представляет отдельный модуль специального вида – **модуль класса**.

Модуль класса: меню **Insert → Class Module**.

Этот модуль имеет такую же структуру, как и стандартный модуль: состоит из двух разделов - объявлений и методов. В первом описываются свойства класса, а во втором - его методы.

**Имя класса** – имя модуля класса.

**Свойства:** процедуры свойств или описания в открытой части раздела (Public) модуля класса.

**Методы:** процедуры и функции (Public) в модуле класса.

Пример.

#### 1. Создание класса.

Программный код в модуле класса с именем **Customer**

```
Public id As Integer
Public Function showsales()
Select Case id
Case 5
showsales = "пять"
Case Else
showsales = "Не пять"
End Select
End Function
```

#### 2. Создания экземпляра (объекта) класса **Customer**, установка свойства и вызов метода.

```
Dim cstMyCust As New Customer
cstMyCust.ID = 5
MsgBox cstMyCust.ShowSales
```

#### 3. Создания двух объектов класса **Customer**

```
Dim cust1 As New Customer
Dim cust2 As New Customer
```

### **5.2. Объектная иерархия объектов приложения**

#### Доступ к объекту

- С указанием полного пути до объекта в иерархической структуре  
Application.Workbooks(1).Worksheets(1).Range("A1").Value = 5
- Без указания пути  
Range("A1").Value = 5  
ActiveSheet.PrintOut

### Использование коллекций

Группы объектов могут быть объединены в коллекции. На коллекцию можно ссылаться как на единое целое.

Два пути указания ссылки на объект в коллекции

1. По имени:

```
Application.Workbooks![Пример Книга1.xls].PrintPreview  
Application.Workbooks("Пример Книга1.xls").PrintPreview
```

2. По индексу в коллекции:

```
Application.Workbooks(1).PrintOut
```

Методы и свойство встроеного динамического класса Collection VBA

Метод	Действие
Add (item, [key], [before], [after])	Добавляет элементы в коллекцию
Remove(key) или Remove(индекс)	Удаляет элементы из коллекции
Item(key) или Item(индекс)	Возвращает значение элемента по ключу или по индексу

item – значение элемента;

key - уникальный ключ;

before, after – индекс или ключ элемента, перед или после которого вставляется элемент.

Метод Item – используется по умолчанию.

Collection(1) – это то же самое, что Collection.Item(1).

**Свойство:** Count – возвращает число элементов в коллекции

### Цикл по элементам в коллекции

Пример.

```
Sub ListDocuments()  
    Dim doc As Document  
    For Each doc In Documents  
        Debug.Print doc.Name  
    Next  
End Sub
```

### Использование объектной переменной

Объектная переменная используется для ссылки на объект.

Сначала описывается объектная переменная, потом указывается **Set** для присвоения объекта переменной.

Пример.

```
Dim rngTest As Excel.Range  
Set rngTest = Application.Workbooks(1).Worksheets(1).Range("A1")  
rngTest.Value = 5
```

1. Описание объектных переменных

```
Dim xlsSht As Excel.Worksheet - раннее связывание  
Dim genObj As Object - позднее связывание
```

2. Установка объектной переменной

Пример.

```
Dim xlsWbk As Excel.Workbook
```

```

Set xlsWbk = Workbooks(1)
xlsWbk.PrintOut
Dim wrdDoc As Document
Set wrdDoc = Documents.Add
wrdDoc.Content.InsertAfter "Hello there."

```

### 3. Nothing

- Определение установлена ли объектная переменная

```

Dim xlsSht As Worksheet
...
If xlsSht Is Nothing Then
    MsgBox "Переменная не указывает на объект"
Else
    MsgBox "Переменная содержит ссылку на объект"
End If

```

Прекращение связи объектной переменной с объектом

```
Set xlsSht = Nothing
```

### 5.3. Работа с объектами

Все объекты можно разделить на 3 группы

1. Объекты, чей класс определен пользователем в одном из модулей класса  
Только спецификатор **New** позволяет создать новый объект для классов, определенных программистом.
2. Объекты родного приложения (Excel, Word...), которому принадлежит проект и которые доступны по умолчанию  
Для создания таких объектов не используются ни спецификатор **New**, ни функция **CreateObject**. Новые объекты, если и создаются, то специальными методами своего класса (например, метод **Add** коллекции **Workbooks**)
3. Active-X и ComAddIns – объекты, в частности объекты других приложений Office при их подключении к исходному приложению.  
Для создания таких объектов используются спецификатор **New** или функция **CreateObject**.

Работая с объектами, Вы работаете с его свойствами и методами.

Синтаксис

```
Объект.свойство, Объект.Метод
```

Пример.

```
MsgBox Application.Workbooks.Count & " open workbooks."
```

**Доступ к свойствам**

```

Dim strDocAuthor As String
Dim wrdDoc As Document
Set wrdDoc = Documents("MyDoc")
application.UserName="Mary"           '- задать свойство объекта
strDocAuthor = application.UserName ' - прочитать свойство объекта

```

**Использование методов**

```
Workbooks(1).PrintOut From:=2, To:=3, Copies:=1
```

**Использование возвращаемых значений метода**

```
Dim wrdDoc As Document
```



```
Set wrdDoc = Application.Documents.Add
wrdDoc.Content.InsertAfter "Hello"
Set d = Documents.Add(Template:="expense.dot")– метод возвращает значение
Documents.Add Template:="expense.dot" – не возвращает значение
```

### Использование With ...End With

Инструкция **With** позволяет указывать объект только один раз для последовательности инструкций. При использовании **With ...End With** программа работает быстрее.

Например, вместо

```
Selection.Font.Bold = True
Selection.Font.Italic = True
Selection.Font.Underline = True
```

МОЖНО ИСПОЛЬЗОВАТЬ:

```
With Selection.Font
.Bold = True
.Italic = True
.Underline = True
End With
```

### 5.4. Автоматизация офисных приложений

Можно использовать автоматизацию (ранее называемую OLE-Automation) для использования объектов одного приложения в другом. Вы можете создать процедуру в WORD, которая будет работать с объектами Microsoft Excel, PowerPoint или Microsoft Access.

Две составляющие автоматизации: Клиент и Сервер. Приложения Office 2010 могут быть как клиентами, так и серверами.

#### Ссылки на объектные библиотеки

При использовании Автоматизации (использования объектов другого приложения) надо сначала установить ссылку на объектную библиотеку и сослаться на объект с первого объекта в объектной иерархии.

Объектная библиотека описывает типы объектов, имеющихся в приложении и методы и свойства объектов этого приложения.

В меню редактора Visual Basic Сервис (Tools) → Ссылки (References...), указать ссылку на необходимую объектную библиотеку (рис. 5.1).

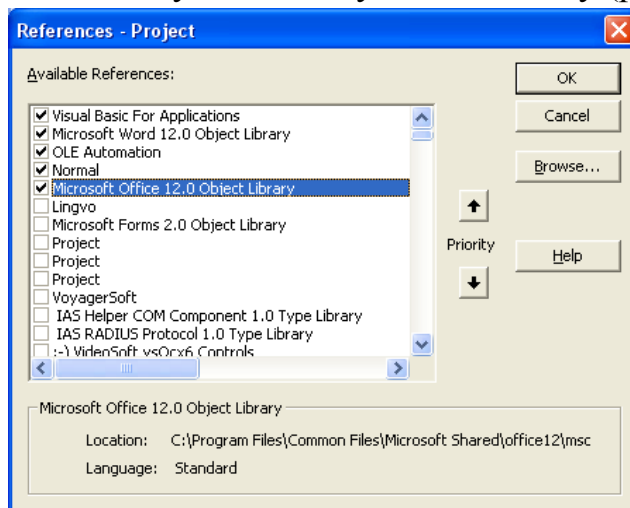
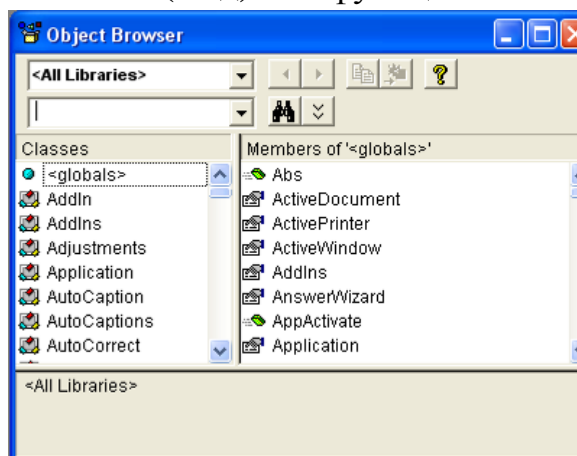


Рис. 5.1 Окно диалога Ссылки

## Использование окна «Просмотр объектов»

Для просмотра информации, хранящейся в объектных библиотеках, используется Окно **Object Browser** (Просмотр объектов) (рис. 5.2), которое можно открыть из меню **View (Вид)** или функциональной клавишей **F2**.



*Рис. 5.2 Просмотр объектов*

Библиотеки, входящие в каркас документа по умолчанию:

**Word (Excel...)** - библиотека, задающая основу документов приложения Word (Excel...). Здесь хранится корневой объект **Application** и все классы объектов, вложенных в корневой объект.

**Office** - библиотека объектов, общих для всех приложений Office 2010.

**Stdole** - библиотека классов для работы с OLE – объектами для реализации Автоматизации.

**VBA** - библиотека классов языка VBA. Здесь хранятся все стандартные функции, встроенные в язык, и многое другое.

**Project** - проект, связанный с документом.

### Ссылки в приложениях

### Использование **CreateObject(Имя\_приложения.Тип\_объекта)**

Функция **CreateObject** создает новый экземпляр объекта и возвращает ссылку на запущенный экземпляр объекта. Если объект может существовать в одном экземпляре (например, PowerPoint), функция вернет запущенный экземпляр объекта.

```
Dim xlsApp As Excel.Application
Dim wrdApp As Word.Application
Dim pptApp As PowerPoint.Application
Set xlsApp = CreateObject ("Excel.Application.9")
Set wrdApp = CreateObject ("Word.Application.9")
Set pptApp = CreateObject ("PowerPoint.Application.9")
```

### Использование **GetObject([pathname] [, class])**

Возвращает ссылку на объект ActiveX (Объект, который может быть открыт для других приложений и средств программирования через интерфейсы программирования), сохраненный в файле.

**Pathname** - Полный путь и имя файла, содержащего объект, который следует загрузить. Если аргумент **pathname** опущен, должен быть указан аргумент **class**.

**Class (имяПриложения.типОбъекта)** - Строка, представляющая класс объекта.

Если аргумент **pathname** опущен, **GetObject** возвращает текущий активный объект указанного типа.

При наличии текущего экземпляра объекта, необходимо использовать функцию **GetObject**.

Пример.

```
Sub TestForApp()
On Error GoTo StartExcel
    Set xlsApp = GetObject("Excel.Application")
Exit Sub
StartExcel:
    If Err.Number = 429 Then
        MsgBox "Будет запущен excel."
        Set xlsApp = CreateObject("Excel.Application")
        xlsApp.Visible = True
    MsgBox "Запустили excel."
    Else
        MsgBox Err.Description
    End If
End Sub
```

### **Примеры управления приложениями**

#### Управление Word - приложением

```
Sub AutomateWord()
' Для тестирования кода - вставьте код в одно из приложений
' Microsoft Excel, PowerPoint или Microsoft Access
' и установите ссылку на Word object library
'Замечание: Если Вы работаете под Windows NT – вы должны
'изменить операторы описания
'Dim wrdApp As Object
'Dim wrdDoc As Object
Dim wrdApp As Word.Application
Dim wrdDoc As Word.Document
Set wrdApp = CreateObject("Word.Application")
wrdApp.Visible = True
Set wrdDoc = wrdApp.Documents.Add
wrdDoc.Content.InsertAfter "Hello World!"
wrdDoc.SaveAs ("C:\MyNewWordDoc.doc")
wrdDoc.Close
wrdApp.Quit
End Sub
```

#### Управление Excel- приложением

```
Sub AutomateExcel()
' Для тестирования кода - вставьте код в одно из приложений
' Word, PowerPoint или Microsoft Access
' и установите ссылку на Excel object library
' Замечание: Если Вы работаете под Windows NT – вы должны
' изменить операторы описания
'Dim xlsApp As Object
Dim xlsApp As Excel.Application
Set xlsApp = CreateObject("Excel.Application")
```

```

xlsApp.Visible = True
xlsApp.Workbooks.Add
xlsApp.Range("A1") = "Hello World!"
xlsApp.Workbooks(1).SaveAs _
    ("C:\MyNewExcelFile.xls")
xlsApp.Quit
End Sub

```

#### Управление PowerPoint- приложением

```

Sub AutomatePowerPoint()
    Dim pptApp As PowerPoint.Application
    Dim pptPres As PowerPoint.Presentation
    Dim pptSlide As PowerPoint.Slide
    Set pptApp = CreateObject("PowerPoint.Application")
    pptApp.Visible = True
    Set pptPres = pptApp.Presentations.Add
    Set pptSlide = pptApp.ActivePresentation.Slides.Add(1, ppLayoutText)
    pptSlide.Shapes("Rectangle 2").TextFrame.TextRange.Text = "Hello"
    pptSlide.Shapes("Rectangle 3").TextFrame.TextRange.Text = _
        "To the entire World!"
    pptApp.Presentations(1).SaveAs _
        ("C:\NewPowerPointPresentation.ppt")
    pptApp.Quit
End Sub

```

#### Управление Microsoft Access - приложением

```

Sub DisplayCatalog()
    Static accApp As Access.Application
    Set accApp = GetObject _
        ("C:\Program Files\Microsoft Office\Office\Samples\Northwind.mdb")
    accApp.Visible = True
    accApp.DoCmd.OpenReport "Alphabetical List of Products", View:=acNormal
    accApp.Quit
End Sub

```

#### Управление Outlook - приложением

```

Sub SendMail()
    Dim olkApp As Outlook.Application
    Dim maliNewMail As MailItem
    Set olkApp = CreateObject("Outlook.Application")
    Set maliNewMail = olkApp.CreateItem(olMailItem)
    With maliNewMail
        .To = "devtrain@microsoft.com"
        .Body = "Please review the proposal I sent you."
        .Send
    End With
End Sub

```

#### **Типичные свойства и методы приложений**

- Свойство Visible – видимость;
- Метод Quit – выход из приложения;
- Метод Activate – активизация приложений;
- Метод Run - запуск макроса;
- Методы Copy and Paste вложенных объектов...

В различных приложениях многие методы совпадают по именам и предназначены для решения стандартных задач, организованы по-разному.

Например, метод **Quit()** в Excel не имеет параметров, он естественно завершает приложение. В Word у этого метода три параметра **SaveChanges**, **Format**, **RouteDocument**, - позволяющие в момент выхода указать, сохранять ли сделанные изменения и их формат, пересылать ли документ всем, кто работает с ним.

### **5.5. Практическое занятие 4: Модели объектов и автоматизация**

На этом занятии вы используете возможности автоматизации (Automation) для работы с другими приложениями Microsoft Office 2010.

#### **Задание 4.1 Использование окна Просмотр объектов**

В этом задании вы используете окно **Object Browser** для изучения различных моделей объектов в Microsoft Office 2010.

➤ Добавление ссылки на объектную библиотеку

1. Откройте редактор Visual Basic в новом документе Microsoft Word.
2. Установите ссылку на Microsoft Excel Object Library (в меню Visual Basic **Сервис (Tools) → Ссылки (References...)**, выбрать Microsoft Excel Object Library и нажать **ОК**)

➤ Просмотр модели объектов

1. Выберите пункт меню **Вид (View) → Просмотр объектов (Object Browser)** или F2.
2. В окне **Object Browser** выберите класс (Classes) **Worksheet**.
3. В «Компонентах» (**Members**) просмотрите три свойства (например, **CodeName**, **Rows**, **Visible...**) и три метода объекта **Worksheet** (например, **Activate**, **Calculate**, **Move...**).

➤ Поиск модели объектов

1. В окне Object Browser найдите все объектные библиотеки и классы, использующие **Range**. Задайте в поле **Поиск текста (Search Text) – Range**.
2. Сколько вариантов **Range** вы видите в панели **Результаты поиска (Search Results)**?

При поиске вы можете найти много вариантов искомого текста. В разных моделях объектов есть свойства, методы, объекты с одинаковыми именами. Вы должны быть уверены, что найденный объект есть тот, что необходим.

3. Сравните свойства и методы класса **Range** в Microsoft Word с классом **Range** в Microsoft Excel. Найдите их в справочной системе для каждого приложения.

#### **Задание 4.2 Работа с объектами**

В этом задании вы напишите код для работы со свойствами и методами объектов.

➤ Использование свойств

1. Откройте новую рабочую книгу Microsoft Excel, затем редактор Visual Basic.
2. Вставьте новый модуль для кода.
3. Создайте новую процедуру.
4. Используйте оператор **With** для вывода некоторых свойств объекта **Application** в окне Отладки (Immediate).  
Используйте свойства `Caption`, `OrganizationName`, `StartupPath`, `UserName`, и `Version`.
5. Измените текст поля состояния, установив свойство **StatusBar** объекта **Application** равным некоторой строке.

```
Sub ExploreProperties()
    With Application
        ' печать некоторых свойств объекта Application
        Debug.Print .Caption
        Debug.Print .OrganizationName
        Debug.Print .StartupPath
        Debug.Print .UserName
        Debug.Print .Version
        .StatusBar = "Hello World!"
    End With
End Sub
```

6. Протестируйте процедуру и проверьте поля состояния в Microsoft Excel. Для просмотра поля состояния активизируйте рабочий лист Microsoft Excel и посмотрите на текст в левом нижнем углу окна приложения.

➤ Использование методов

1. Создайте новую процедуру.
2. Опишите объектную переменную типа **Worksheet**.
3. Присвойте этой переменной значение, равное третьей странице в этой книге.
4. Используйте метод **Activate** объекта **worksheet** для активизации этой страницы.
5. Используйте свойство **Range** объекта **worksheet** для задания значения ячейке A1.
6. Используйте метод **PrintPreview** для объекта **worksheet**.
7. Используйте метод **Help** объекта **Application** для вывода справки.

```
Sub ExploreMethods()
    Dim wksCurrent As Worksheet
    '
    Set wksCurrent = Worksheets("Лист3")
    wksCurrent.Activate
    wksCurrent.Range("A1") = "Hello World!"
    wksCurrent.PrintPreview
    Application.Help
End Sub
```

8. Протестируйте процедуру.

### Задание 4.3 Работа с объектными переменными

В этом задании вы будете работать с объектными переменными.

#### ➤ Описание объектных переменных

1. Создайте новый документ Word.
2. В проект этого документа добавьте новый модуль и процедуру типа **Sub**.
3. Опишите переменную типа **Object**.  
Переменная описанная как **Object** обладает поздним связыванием с объектом.
4. Опишите вторую переменную типа **Document**.  
Переменная, описанная как объект определенного типа обладает ранним связыванием с объектом.

#### ➤ Использование объектных переменных

1. Введите имя переменной, связанной с документом, за ней поставьте оператор (.). Что вы увидите?  
Так как было использовано раннее связывание, вы увидите список доступных методов и свойств.
2. Используйте оператор **Set** для связи переменной с активным документом.
3. Задайте вывод свойства **Name** объекта **Document** в окне Отладки (Immediate).
4. Введите имя переменной, описанной как **Object**, за ней поставьте оператор (.). Что вы увидите?  
Так как было использовано позднее связывание, вы не увидите никаких методов и свойств.
5. Используйте оператор **Set** для связи переменной с активным документом.
6. Используйте свойство **Name** объектной переменной для вывода его в окно Отладки (Immediate).

```
Sub ObjectVariables()  
    Dim objMyObject As Object  
    Dim docMyDocument As Document  
    Set docMyDocument = Application.ActiveDocument  
    Debug.Print docMyDocument.Name  
    Set objMyObject = Application.ActiveDocument  
    Debug.Print objMyObject.Name  
End Sub
```

#### ➤ Протестируйте ваш код:

Запустите процедуры и посмотрите окно Отладки (Immediate)

### Задание 4.4 Работа с несколькими объектными библиотеками

В этом задании вы поработаете с несколькими библиотеками объектов.

#### ➤ Описание объектов

1. Создайте новую презентацию Microsoft Power Point.
2. В редакторе Visual Basic, создайте новую процедуру типа **Sub**.
3. Опишите переменную типа **Excel.Worksheet**.
4. Выберите пункт меню **Отладка (Debug) → Компилировать Project (Compile Project)**.

Так как у вас нет ссылки на библиотеку объектов Microsoft Excel, вы получите сообщение об ошибке.

5. Задайте ссылку на библиотеку объектов Microsoft Excel.

6. Откомпилируйте проект снова.

Сообщений об ошибке не будет.

➤ Работа с несколькими объектными библиотеками

1. Задайте ссылку на библиотеку объектов Word.

2. Опишите переменную **rngWord** типа **Range**.

Вы создали переменную типа Range в библиотеке объектов Microsoft Excel.

3. Введите имя **rngWord**, за которым следует оператор точка (.).

Вы увидите свойства и методы для библиотеки Microsoft Excel объект **Range**.

➤ Изменение приоритетов модели объектов

1. Выберите пункт меню Сервис (Tools) → Ссылки (References).

2. Используйте кнопки Приоритет (Priority) для перемещения объектной библиотеки Word выше над объектной библиотекой Microsoft Excel.

Это приведет к тому, что описание типа **Range** будет связано с объектной библиотекой Word.

3. Введите имя переменной **rngWord**, за которым идет оператор точка (.).

Вы увидите свойства и методы **Range** из библиотеки Word.

➤ Описание переменной типа Range с указанием библиотеки

1. Измените описание **rngWord** на тип **Word.Range**.

2. Опишите переменную **rngExcel** типа **Excel.Range**.

3. Введите **rngWord**, за ним оператор точка ( . ).

Вы увидите свойства и методы объекта **Range** из библиотеки Word.

4. Введите **rngExcel**, за ним оператор точка ( . ).

Вы увидите свойства и методы объекта **Range** из библиотеки Excel

#### Задание 4.5 Использование автоматизации

В этом задании вы используете автоматизацию приложений Office 2010.

➤ Автоматизация Microsoft PowerPoint

1. Создайте новый документ Microsoft Excel или Word.

2. В редакторе Visual Basic, задайте ссылку на библиотеку PowerPoint.

3. Создайте новый модуль и новую процедуру типа **Sub**.

4. Опишите переменную типа **PowerPoint.Application**. Используйте функцию **CreateObject** для связи переменной с новой объектом PowerPoint.

5. Выведите сообщение о запуске PowerPoint.

6. Используйте метод **Quit** объекта **Application** для завершения работы с PowerPoint.

```
Sub AutomatePowerPoint()
```

```
Dim pptApp As PowerPoint.Application
```



```

Set pptApp = CreateObject("PowerPoint.Application")
MsgBox "Запуск PowerPoint!"
    pptApp.Quit
End sub

```

➤ Протестируйте ваш код

1. Запустите вашу процедуру.
2. При появлении окна с сообщением, найдите PowerPoint. Видите ли вы его?

Хотя PowerPoint запущен. Он не видим.

3. Закройте окно сообщения.
4. Перед кодом с выводом сообщения, установите свойство **Visible** объекта **Application** для PowerPoint равным True (pptApp.Visible = msoTrue)
5. Запустите процедуру снова. Перед появлением окна сообщения PowerPoint станет видим.

➤ Добавление нового документа

➤ В имеющуюся процедуру добавьте операторы для создания и сохранения презентации PowerPoint.

```

Sub AutomatePowerPoint()
Dim pptApp As PowerPoint.Application
Dim pptDoc As PowerPoint.Presentation
' создание экземпляра PowerPoint
Set pptApp = CreateObject(«PowerPoint.Application»)
' сообщение для пользователя
MsgBox "Запуск PowerPoint!"
pptApp.Visible = msoTrue      ' сделать PowerPoint видимым
' создание новой презентации
Set pptDoc = pptApp.Presentations.Add
' сохранение презентации
pptDoc.SaveAs («C:\NewPPTFile.ppt»)
pptApp.Quit      ' закрытие PowerPoint
' сообщение для пользователя
    MsgBox «PowerPoint закрыт!»
End Sub

```

## 6. Использование Microsoft Excel объектов

### 6.1. Объектная модель Microsoft Excel

Объекты Microsoft Excel

Каждый компонент в Microsoft Excel может быть представлен как объект VBA.

Например, Рабочая книга → Workbook

Рабочий лист → Worksheet

Лист → Sheet

Набор ячеек → Range

Диаграмма → Chart

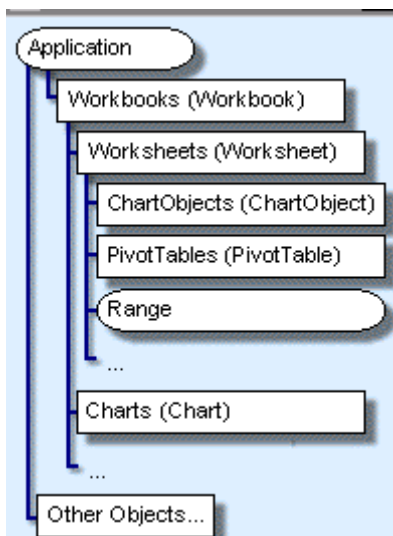


Рис 6.1. Объектная модель Microsoft Excel

Наиболее часто используемые объекты

Объект	Описание
Application	Корневой объект, в который вложены все остальные объекты
Workbook	Рабочая книга, в которой хранятся листы с данными
Worksheet	Рабочий лист - непосредственно документ для работы с данными
Range	Область ячеек
Chart	Диаграмма (графическое представление данных)
PivotTable	Сводная таблица (таблица для быстрого подведения итогов или объединения больших объемов данных)

### Ориентация в объектной модели Microsoft Excel

Доступ к объектам Microsoft Excel из Excel-приложения

- полный путь:

Application.Workbooks("MyBook.xls").Worksheets("Sales").Range("A1").Select

- краткий путь:

Range("A1").Select

Можно использовать свойства объекта **Application** ActiveCell, ActiveChart, ActiveSheet, ActiveWorkbook...

Доступ к объектам Microsoft Excel из других приложений:

```
Sub AutomateExcel()  
    ' поставить ссылку на библиотеку  
    Dim xlApp as Excel.Application  
    Set xlApp = CreateObject("Excel.Application")  
    xlApp.Visible = True  
End Sub
```

## 6.2. Использование объекта *Workbook* и коллекции *Workbooks*

### Открытие

- открытие существующей рабочей книги (метод **Open** объекта **Workbooks**)

```
Sub OpenAnalysis()  
    Workbooks.Open "C:\My Documents\Analysis.xls"  
End Sub
```

- создание новой рабочей книги (метод **Add** объекта **Workbooks**)

```
Sub AddNewWorkbook ()  
    Application.Workbooks.Add  
End Sub
```

### Сохранение

- сохранение книги с текущим именем книги (метод **Save** объекта **Workbook**)

```
Sub SaveAll()  
    Dim wkBook As Workbook  
    For Each wkBook In Application.Workbooks  
        wkBook.Save  
    Next wkBook  
End Sub
```

- сохранение книги с новым именем (метод **SaveAs** объекта **Workbook**)

```
Sub SaveAsConvert()  
    ActiveWorkbook.SaveAs "Convert.XLS", _  
        FileFormat:=xlWKS    'XL 5.0 Format  
End Sub
```

### Закрытие

- метод **Close** объекта **Workbook**

```
Sub CacheClose()  
    ActiveWorkbook.Close SaveChanges:=True, _  
        FileName:= "C:\My Documents\CACHE.XLS"  
End Sub
```

### События **Workbook**

#### 1. событие **Open**

```
Private Sub Workbook_Open()  
    Windows(1).DisplayWorkbookTabs = False  
End Sub
```

#### 2. событие **Activate**

```
Private Sub Workbook_Activate()  
    Application.CommandBars("Web").Visible = True  
End Sub
```

### 3. событие Deactivate

```
Private Sub Workbook_Deactivate()  
    Application.CommandBars("Web").Visible = False  
End Sub
```

### 4. событие BeforeClose

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)  
    ActiveWorkbook.BuiltinDocumentProperties("Author") = "Jean Selva"  
End Sub
```

### 5. событие BeforePrint

```
Private Sub Workbook_BeforePrint (Cancel As Boolean)  
    If ActiveWorkbook.Saved = False Then  
        ' Saved – true, если не было изменений после сохранения документа  
        MsgBox "Пожалуйста, сохраните книгу перед печатью"  
        Cancel = True  
        Exit Sub  
    End If  
End Sub
```

### 6. событие NewSheet

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)  
    ' при добавлении листа, он перемещается в конец книги  
    Sh.Move After:=Sheets(Sheets.Count)  
End Sub
```

## 6.3. Использование Worksheet

### Select (выделение)

Для выделения рабочего листа используется метод **Activate** объекта **Worksheet**

```
Sub ActivateSalesInfo()  
    ' если в книге есть лист с именем "SalesInfo", сделать его активным  
    ' в противном случае, добавить лист с этим именем  
    Dim sh as Worksheet  
    For Each sh In Worksheets  
        If sh.Name ="SalesInfo" Then  
            sh.Activate  
            Exit Sub  
        End If  
    Next  
    ActiveWorkbook.Worksheets.Add  
    ActiveSheet.Name = "SalesInfo"  
End Sub
```

### Вставка данных в Worksheet

#### Свойство Range

- 1) Range("A1").Value = "Треугольник"
- 2) Range("Profit").Formula = "=SUM(A1:A3)"

#### Пример.

```
Sub SortMyRange()  
    Dim rngNames As Range  
    Set rngNames = Range("A1", "A25")  
    rngNames.Sort Key1:=Range("A1"), Order1:=xlDescending  
End Sub
```

## Свойство **Offset**

Свойство **Offset** объекта **Range** имеет два параметра **RowOffset** и **ColumnOffset** – смещение по строкам и столбцам, и возвращает новый объект, отстоящий от прежнего на заданное расстояние.

```
Sub ValidateRange()  
    Dim intCheckCells As Integer  
    Worksheets("Лист1").Activate  
    Range("A1").Activate  
    For intCheckCells = 1 To 10  
        If Not IsNumeric(ActiveCell) Then  
            MsgBox "Здесь не число!"  
            Exit Sub  
        End If  
        ActiveCell.Offset(1, 0).Activate  
    Next  
End Sub
```

## Свойство **Cells**

Свойство **Cells** объекта **Range** возвращает объект **Range**. Позволяет работать как со всеми ячейками листа, так и с единичными ячейками.

```
Sub MultiplicationTable()  
    Dim intRow As Integer, intCol As Integer  
    For intRow = 1 To 10  
        For intCol = 1 To 10  
            Cells(intRow, intCol).Value = intRow * intCol  
        Next intCol  
    Next intRow  
End Sub
```

## События **Worksheet**

### 1. событие **Activate**

```
Private Sub Worksheet_Activate()  
    Range("a1:a10").Sort Key1:=Range("a1"), Order1:=xlAscending  
End Sub
```

### 2. событие **Calculate**

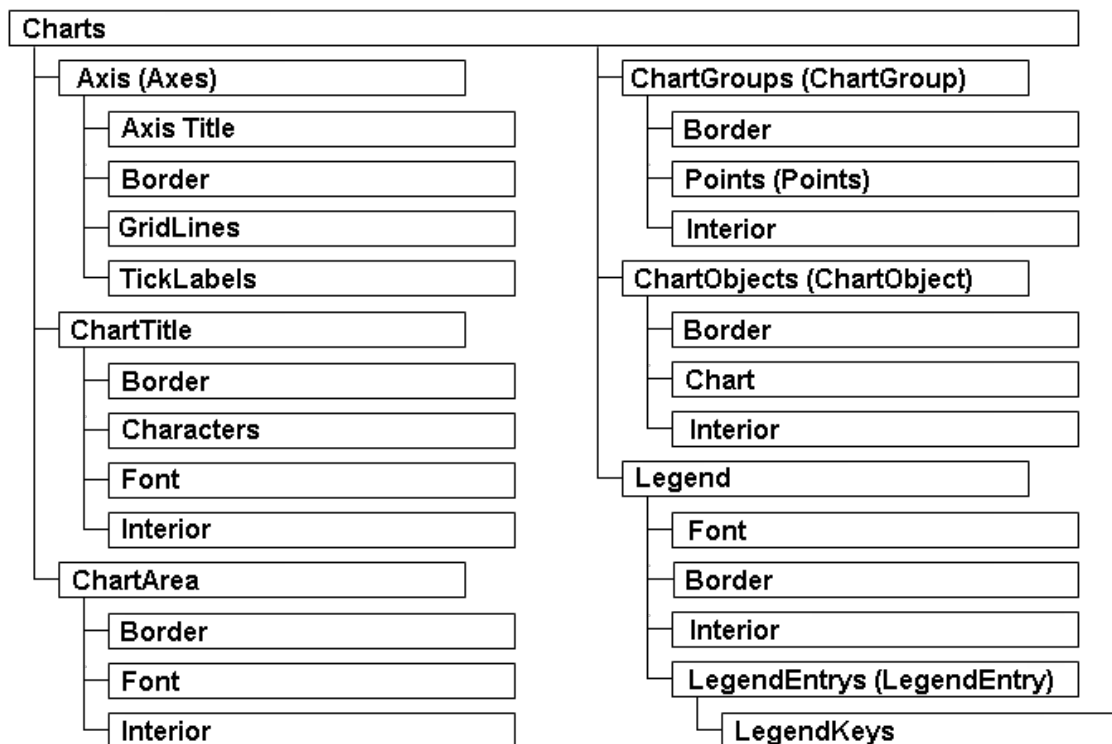
```
Private Sub Worksheet_Calculate()  
    Columns("A:F").AutoFit  
End Sub
```

### 3. событие **Deactivate**

```
Private Sub Worksheet_Deactivate()  
    Dim intResponse As Integer  
    intResponse = MsgBox("Запускать проверку?", vbYesNo)  
    If intResponse = vbYes Then  
        ' [запускается код проверки]  
    End If  
End Sub
```

## 6.4. Создание **Charts** (диаграмм)

На рис. 6.2 представлена иерархия объектов, которые используются при работе с отдельными диаграммами, а также с диаграммами, размещенными на листах.



*Рис. 6.2 Иерархия объектов для работы с диаграммами*

На самом верху иерархии расположена Коллекция (**Charts**), которая содержит все диаграммы рабочей книги, находящиеся на отдельных листах.

Для работы с диаграммами, внедренными в рабочие листы, необходимо использовать коллекцию **ChartObjects**, которая содержит все объекты **ChartObject**, являющиеся внедренными диаграммами конкретного рабочего листа. Для быстрой программной реализации диаграммы можно воспользоваться методом **ChartWizard** объекта **Chart**. Его использование не требует отдельного задания свойства диаграммы.

### Коллекция Charts

Коллекция **Charts** содержит все листы с диаграммами конкретной рабочей книги. Каждый лист диаграммы представляет собой объект **Chart**. Эта коллекция не содержит диаграммы, внедренные в рабочий лист или лист диалога.

#### Методы коллекции Charts

#### Метод Add

Метод Add добавляет пустой лист диаграммы. Синтаксис этого метода:

Expression.Add(Before, After, Count)

Expression - выражение, которое возвращает объект **Charts**.

Before - необязательный параметр типа Variant. Данный параметр определяет лист рабочей книги, перед которым будет вставлен добавляемый лист диаграммы.

After - необязательный параметр типа Variant. Предназначен для определения листа рабочей книги, после которого будет вставлен добавляемый лист диаграммы. В том случае, если оба параметра Before и After отсутствуют, лист диаграммы будет вставлен перед активным листом.

Count - необязательный параметр типа Variant. Предназначен для определения количества добавляемых листов. По умолчанию данный параметр имеет значение равное 1.

Пример фрагмента программы, реализующей добавление в активную рабочую книгу листа диаграммы, размещаемого после листа с именем Лист1.

```
ActiveWorkbook.Charts.Add After:=Worksheets("Лист1")
```

### Метод Delete

Метод Delete позволяет удалить все листы диаграмм из коллекции **Charts**.

### Объект Chart

Объект **Chart** представляет собой диаграмму, расположенную в рабочей книге на отдельном листе диаграммы. Все объекты **Chart** являются элементами коллекции **Charts**.

#### Свойства объекта Chart

Ниже рассмотрены некоторые свойства объекта **Chart**, позволяющие управлять внешним видом диаграммы.

Свойство **Legend** возвращает ссылку на объект Legend, который представляет собой легенду диаграммы.

Свойство **ChartArea** возвращает ссылку на объект ChartArea, который предоставляет возможность работать с областью диаграммы.

Свойство **ChartTitle** возвращает ссылку на объект ChartTitle, представляющий заголовок диаграммы.

Свойство **ChartType** определяет тип диаграммы. Значением этого свойства может быть значение одной из констант, представленных в таблице, например, <http://www.taurion.ru/excel/pril1/32>

Свойство **HasAxis** определяет тип осей, присутствующих на диаграмме. Синтаксис данного свойства представлен ниже:

```
Expression.HasAxis(Index1, Index2)
```

Expression Выражение, возвращающее объект Chart

Index1 Необязательный параметр типа Variant. Определяет тип осей и может иметь одно из следующих значений:

XlCategory – ось категорий,

XlValue – ось значений,

XlSeriesAxis – ось рядов данных. Значение этой константы имеет смысл только для линейных диаграмм

Index2 Необязательный параметр типа Variant. Определяет группу осей и может иметь одно из двух значений: xlPrimary или xlSecondary. Трехмерные диаграммы имеют только одну группу осей

Свойство **HasDataTable** определяет присутствие (значение True) или отсутствие (значение False) таблицы данных на диаграмме.

Свойство **HasLegend** задает наличие (значение True) или отсутствие (значение False) легенды на диаграмме.

Свойство **HasTitle** определяет наличие (значение True) или отсутствие (значение False) заголовка и осей диаграммы.

Свойство **Rotation** задает угол отображения трехмерных диаграмм. Его значение измеряется в градусах в диапазоне от 0 до 360 (для трехмерной линейчатой диаграммы это значение должно лежать в пределах от 0 до 44). По умолчанию для новой диаграммы значение свойства **Rotation** равно 20.

#### Методы объекта Chart

Из существующего набора методов объекта **Chart** мы рассмотрим здесь только один из них, наиболее полезный – **ChartWizard**. Этот метод позволяет быстро задать параметры диаграммы без необходимости определения каждого ее свойства в отдельности. Синтаксис этого метода <http://www.taurion.ru/excel/pril1/33>.

#### События объекта Chart

Объект **Chart**, как и большинство других объектов, имеет набор событий, включающих **Activate**, **Deactivate**, **MouseDown**, **MouseUp** и т. п.

#### Событие **Calculate**

Для объекта **Chart** событие **Calculate** происходит после того, как изменяются данные, лежащие в основе диаграммы.

### **6.5. Создание сводных таблиц**

При помощи кубов OLAP (многомерные таблицы, в которых вместо стандартных двух измерений их может быть больше) просто получать ответы на такие вопросы, как *"сколько товаров такого-то типа было продано в четвертом квартале прошлого года в Северо-Западном регионе через региональных дистрибьюторов"*.

Обычно для описания измерений в кубе используется термин "в разрезе". Например, отделу маркетинга может быть нужна информация во временном разрезе, в региональном разрезе, в разрезе типов продукта, в разрезе каналов продаж и т. п.

Для работы с такими кубами в Excel встроен специальный клиент – **PivotTable (Сводная таблица)** доступен на вкладке **Вставка** в группе **Таблицы**.

Необходимо отметить, что он умеет работать не только с кубами OLAP, но и с обычными данными в таблицах Excel или в базах данных.

Сводная таблица и объект **PivotTable** — это программные продукты фирмы **Panorama Software**, которые были приобретены **Microsoft** и интегрированы в Excel. Поэтому работа с объектом **PivotTable** несколько отличается от работы с другими объектами Excel. Поэтому рекомендуется для получения подсказок рекомендуется использовать макрорекордер.

#### **Порядок создания сводной таблицы**

При работе со сводной таблицей необходимо:

1. Создать объект **PivotCache**, который будет представлять собой набор записей, полученных с источника. Для каждого объекта **PivotTable** можно использовать только один объект **PivotCache**. Создание объекта **PivotCache** производится при помощи метода **Add()** коллекции **PivotCaches**:

```
Dim PC1 As PivotCache
```

```
Set PC1 = ActiveWorkbook.PivotCaches.Add(xlExternal)
```



## 2. Настроить параметры объекта **PivotCache**.

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

`TableDestination`— единственный обязательный параметр. Принимает объект **Range**, в верхний левый угол которого будет помещена сводная таблица.

`TableName`— имя сводной таблицы. Если не указано, то автоматически сгенерируется имя типа "СводнаяТаблица1".

`ReadData`— если установить в `True`, то все содержимое куба будет автоматически помещено в кэш. С этим параметром нужно быть очень осторожным, поскольку неправильное его применение может резко увеличить нагрузку на клиента.

`DefaultVersion`— это свойство обычно не указывается. Позволяет определить версию создаваемой сводной таблицы. По умолчанию задается наиболее свежая версия.

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

```
PC1.CreatePivotTableRange("A1")
```

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

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

Часто требуется расположить данные в сводной таблице программным образом.

Эта операция производится при помощи объекта **CubeField**. Главное свойство этого объекта — `Orientation`, оно определяет, где будет находиться то или иное поле.

Например, помещаем измерение `Customers` в область столбцов:

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

измерение `Time` — в область строк:

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

измерение `Product`— в область страницы:

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

и, наконец, показатель `UnitSales` (число для анализа):

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

Модификация созданной сводной таблицы (`PivotTable`)

Пример.

```
Sub CreatePivotTable()  
  With Sheet1  
    .PivotTableWizard SourceType:=xlDatabase, SourceData:= _ "Sales!R1C1:R541C7",  
TableDestination:="", TableName:="Pivot1"  
    .PivotTables("Pivot1").AddFields RowFields:="Period", _  
    ColumnFields:="Product",PageFields:=Array("Continent", "Category")  
  End With  
End Sub
```

Группировка данных

```
Sub GroupQuarterly (pvtTable As PivotTable)  
  Dim rngGroup as Range  
  Set rngGroup = pvtTable.PivotFields("Date").DataRange  
  rngGroup.Cells(1).Group _  
    periods:=Array(False,False,False,False,False,True,False)  
  ' ARRAY (Seconds, Minutes, Hours, Days, Months, Quarters, Years)  
End Sub
```

Обновление данных сводной таблицы

```
pvtTable.RefreshTable
```

## ***6.6. Практическое занятие 5: Работа с Microsoft Excel***

### **Задание 5.1 Создание рабочей книги**

В этом задании вы используете автоматизацию для вставки и форматирования данных в рабочей книге.

1. В Microsoft Word создайте новый модуль.
2. Напишите процедуру, которая создает новую книгу в Microsoft Excel, вставляет текст в ячейку A1, форматирует ячейку (полужирный шрифт, подчеркивание и шрифт Tahoma).
3. Не забудьте установить ссылку на библиотеку объектов Excel.

Возможный пример решения:

```
Sub WrkBk()  
  Static appXl As Object  
  Set appXl = CreateObject(«Excel.Application»)  
  With appXl  
    .Workbooks.Add  
    .Visible = True  
    With .Range(«A1»)  
      .Value = «Привет»  
      .Font.Name = «Tahoma»  
      .Font.Bold = True  
      .Font.Underline = True  
    End With  
  End With  
End Sub
```

### **Задание 5.2 Построение диаграммы**

Требуется

Добавить на рабочий лист диаграмму (типа гистограмма), используя диапазон значений, выделенных пользователем.

## Решение

1. Объявите процедуру:  
Public Sub Diagramma()
2. Объявите переменную для хранения ссылки на диаграмму:  
Dim obj\_Chart As Chart
3. И для хранения ссылки на диапазон входных значений:  
Dim obj\_Range As Range  
Set obj\_Range = Selection
4. Добавляем новую диаграмму:  
ActiveSheet.Shapes.AddChart.Select  
Set obj\_Chart = ActiveChart
5. Укажите исходные данные для диаграммы:  
obj\_Chart.SetSourceData \_  
Source:=obj\_Range, \_  
PlotBy:=xlRows
6. Установите требуемый тип (гистограмма) для диаграммы:  
obj\_Chart.ChartType = xlColumnClustered  
End Sub

### **Задание 5.3 Модификация диаграммы**

1. Откройте файл Ex2.xls в папке \Lab06.
2. Напишите код для цифрового счетчика для изменения угла возвышения, из которого видна диаграмма.
3. Напишите код для командной кнопки восстановления характеристик диаграммы, восстановление свойств Elevation (Возвышение), Rotation (Поворот) и Perspective (Перспектива).

Возможный вариант решения:

```
Private Sub CommandButton1_Click()  
    ' Текст программы обработки события – нажатие кнопки Reset Chart  
    ' Эта процедура восстанавливает характеристики диаграммы  
    Dim chtSales As ChartObject  
        Range(«A1»).Select  
        Set chtSales = Sheet1.ChartObjects(1)  
            With chtSales.Chart  
                .Elevation = 15  
                .Rotation = 20  
                .Perspective = 30  
            End With  
    End Sub
```

```
Private Sub SpinButton1_SpinDown()  
    ' Текст программы обработки события – нажата стрелка вниз  
    ' Свойство Elevation диаграммы может принимать значения от -90 до 90  
    Dim chtSales As ChartObject  
        Range(«A1»).Select  
            Set chtSales = Sheet1.ChartObjects(1)  
            If (chtSales.Chart.Elevation - 10) >= -90 Then  
                chtSales.Chart.Elevation = chtSales.Chart.Elevation - 10  
            End If  
    End Sub
```

```

Private Sub SpinButton1_SpinUp()
' Текст программы обработки события – нажата стрелка вверх
' Свойство Elevation диаграммы может принимать значения от –90 до 90
Dim chtSales As ChartObject
Range(«A1»).Select
Set chtSales = Sheet1.ChartObjects(1)
If (chtSales.Chart.Elevation + 10) <= 90 Then
chtSales.Chart.Elevation = chtSales.Chart.Elevation + 10
End If
End Sub

```

#### 4. Протестируйте код для диаграммы

##### Задание 5.4 Построение сводной таблицы

В данном задании вы программно создадите сводную таблицу с требуемыми полями.

Подготовительное действие: импортируйте в новую книгу (начиная с ячейки A1) таблицу **Products** базы данных **Nwind**.

Добавьте модуль в созданную книгу и создайте в нем процедуру CreatePivot().

В теле процедуры добавьте код, создающий на основе таблицы **Product** сводную таблицу. Для этого выполните следующие действия:

#### 1. Объявите две объектные переменные - **PivotCache** и **PivotTable**:

```

Dim PTCach As PivotCache
Dim PT As PivotTable

```

#### 2. Создайте объект **PivotCache**:

```

Set PTCach = ActiveWorkbook.PivotCaches.Create( _
SourceTypes:=xlDatabase, _
SourceData:=Range("A1").CurrentRegion)

```

#### 3. Добавьте новый лист в книгу, на котором будет строиться сводная таблица:

```
Worksheets.Add
```

#### 4. Создайте сводную таблицу:

```

Set PT = ActiveSheet.PivotTables.Add( _
PivotCache:=PTCach, _
TableDestination:=Range("A3"))

```

#### 5. Расположите данные в требуемых полях сводной таблицы:

```

With PT
.PivotFields("ProductID").Orientation = xlPageField
.PivotFields("QuantityPerUnit").Orientation = xlColumnField
.PivotFields("ProductName").Orientation = xlRowField
.PivotFields("UnitPrice").Orientation = xlDataField
.DisplayFieldCaptions = False
End With

```

#### 6. Откройте лист с исходной таблицей и запустите процедуру CreatePivot().

#### 7. Внесите дополнения в таблицу – добавьте вычисляемое поле *Заказать товар*, вычисляемое как разница между столбцами **ReorderLevel** (**Минимальный запас**) и **UnitsInStock** (**На складе**). В этот столбец

помещается информация о количестве товара в штуках, которое нужно срочно заказать.

Вычисляемое поле можно добавить следующим образом:

```
.CalculatedFields.Add "Zakaz", "=ReorderLevel-UnitsInStock"  
.PivotFields("Zakaz").Orientation = xlDataField
```

### Задание 5.5 Группировка данных в сводной таблице

В этом задании вы создадите сводную таблицу с помощью мастера и добавите код для модификации сводной таблицы.

#### ➤ Создание сводной таблицы

1. Откройте файл Ex3.xls в папке \Lab06. Используйте данные на странице Sales Data для создания сводной таблицы на отдельной странице (вкладка **Вставка** → **Сводная таблица**).
2. Создайте (перетащите)
  - В фильтр отчета поля Continent и Category
  - В область строк поле Period
  - В область столбцов поле Product
  - В область данных (значения) поле Profits

Получится ориентировочно следующий вид сводной таблицы:

	A	B	C	Денежный формат	E
1	Continent	Asia			
2	Category	Accessories			
3					
4	Сумма по полю Profit	Product			
5	Period	Gloves	Helmet	Pump	Общий итог
6	20/01/94	85197	56788	63991	205976
7	20/02/94	40198	30161	96660	167019
8	20/03/94	81761	77355	78490	237606
9	20/04/94	74068	51549	78122	203739
10	20/05/94	91590	68939	31716	192245
11	20/06/94	66174	87692	16650	170516
12	20/07/94	47886	23959	34359	106204
13	20/08/94	22148	61524	64805	148477
14	20/09/94	52575	65186	41506	159267
15	20/10/94	43243	19378	807	63428
16	20/11/94	32349	58874	41477	132700
17	20/12/94	98980	53002	16592	168574
18	Общий итог	736169	654407	565175	1955751

#### ➤ Управление сводной таблицей

1. Задайте имя листа со сводной таблицей (например, Сводная таблица)
2. Добавьте объект флажок () для страницы со сводной таблицей (Вкладка **Разработчик** → группа **Элементы управления** → кнопка **Вставить**).
3. Напишите код для события **Click** флажка – группировка данных по кварталам.

```
Private Sub CheckBox1_Click()  
    Dim pvtTable As PivotTable, rng As Range  
    Range("A1").Select  
    Set pvtTable = Worksheets("Сводная").PivotTables(1)  
    Set rng = pvtTable.PivotFields("Period").DataRange.Rows(1)  
    If CheckBox1.Value = True Then  
rng.Group Periods:=Array(False, False, False, False, False, True, False)  
    Else  
rng.Ungroup
```

End If

End Sub

4. Протестируйте флажок, чтобы убедиться в правильности группировки данных.

#### **Задание 5.6 Работа с событиями**

В этом задании вы напишите код для события **Deactivate** объекта **Worksheet** с данными (Sales Data) - обновление сводной таблицы.

- Написание процедуры обработки событий

```
Private Sub Worksheet_Deactivate()
```

```
Worksheets("Сводная").PivotTables(1).RefreshTable
```

```
End Sub
```

- Тестирование процедуры обработки событий
- Измените данные на странице Sales Data.
- Переключитесь на сводную таблицу для того, чтобы увидеть изменения.

## 7. Использование Microsoft Word объектов

### 7.1. Объектная модель Microsoft Word

#### Объекты Microsoft Word

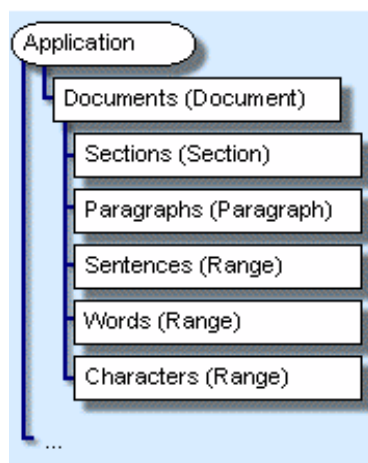


Рис 7.1. Объектная модель

#### Ориентация в объектной модели Microsoft Word

- из Word

```
Application.Documents("doc1.doc").Paragraphs(1).Style = wdStyleHeading1
```

- Автоматизация (из других приложений)

```
Dim wrdApp As Word.Application
Set wrdApp = CreateObject("Word.Application")
wrdApp.Documents.Open FileName:= "c:\myDoc.doc"
```

#### Использование событий

##### События документа

События	Описание
New	Происходит, когда создается новый документ на базе шаблона (только для шаблона)
Open	Происходит, когда открывается документ.
Close	Происходит, когда закрывается документ.

1. Private Sub Document\_New()  
    frmFormLetter.Show  
End Sub
2. Private Sub Document\_Open()  
    Dim rngCurrent As Range  
    Set rngCurrent = ActiveDocument.Content  
    With rngCurrent  
        .Collapse wdCollapseEnd  
        .InsertDateTime "MM/dd/yy HH:MM:SS", False  
    End With  
    Set rngCurrent = ActiveDocument.Content  
    With rngCurrent  
        .InsertParagraphAfter  
        .Collapse wdCollapseEnd  
        .Select  
    End With  
End Sub

## События Application

<b>События</b>	<b>Описание</b>
DocumentChange	Происходит, когда документ создается, открывается или другой документ становится активным.
Quit	Происходит, когда завершается работа с Word.

## Автоматизация

<b>Имя автоматизации</b>	<b>Запускается</b>
AutoExec	При запуске приложения Word.
AutoExit	При закрытии приложения Word.
AutoNew	Каждый раз при создании нового документа.
AutoOpen	Каждый раз при открытии существующего документа.
AutoClose	Каждый раз при закрытии документа

Порядок событий:

1. Автоматизация в документах и шаблонах
2. События шаблона
3. События документа

## **7.2. Работа с Word документами**

### **Открытие, создание**

#### Создание нового документа

На базе шаблона Normal.dot

```
Documents.Add
```

На базе своего шаблона

```
Set wrdDoc = Documents.Add (Template:="C:\MyTemplate.dot")
```

```
With wrdDoc
```

```
    .Content.Font.Name = "Arial"
```

```
    .SaveAs FileName:="Sample.doc"
```

```
End With
```

Для определения имени шаблона

```
Msgbox ActiveDocument.AttachedTemplate
```

#### Открытие существующего документа

```
Documents.Open FileName:="C:\MyFolder\MyDoc.doc"
```

Использование окна диалога:

```
Application.Dialogs(wdDialogFileOpen).Show
```

Использование окна диалога:

```
Public Sub Ex_Dialogs()
```

```
'Application.Dialogs(wdDialogFileOpen).Show
```

```
Set dlg = Dialogs(wdDialogFileOpen)
```

```
If dlg.Display = -1 Then
```

```
    Documents.Open FileName:=dlg.Name
```

```
End If
```

```
End Sub
```



### Определение, открыт ли документ

```
For Each wrdDoc In Documents
  If UCase(wrdDoc.Name) = "SAMPLE.DOC" Then
    wrdDoc.Activate
    blnDocfound = True
  End If
Next wrdDoc
If blnDocfound = False Then
  Documents.Open FileName:="c:\my documents\sample.doc"
End If
```

### **Сохранение, закрытие**

#### Сохранение документа

- **существующего:**  
Documents("Sales.doc").Save
- **НОВОГО**  
Set wrdDoc = Documents.Add  
wrdDoc.SaveAs FileName:="Temp.doc"

#### Закрытие документа

- **всех открытых**  
Documents.Close SaveChanges:=wdDoNotSaveChanges
- **одного открытого**  
Documents("Sales.doc").Close SaveChanges:=wdSaveChanges

### **Печать документа**

Печать 3-й, 4-й и 5-й страницы документа:

```
ActiveDocument.PrintOut Range:=wdPrintFromTo, From:="3", To:="5"
```

Установка параметров печати:

```
Options.PrintHiddenText = True
ActiveDocument.PrintOut
```

Установка ориентации страницы:

```
ActiveDocument.PageSetup.Orientation = wdOrientLandscape
ActiveDocument.PrintOut
```

## **7.3. Работа с частями (областями) документа**

### **Части текста. Структуризация**

Некоторые коллекции объектной модели Word

<b>Имя коллекции</b>	<b>Возвращаемый тип объекта</b>
Words	Range
Characters	Range
Sentences	Range
Paragraphs	Paragraph
Sections	Section

Пример.

```
Dim rngTest As Range
set rngTest = ActiveDocument.Sentences(1)
rngTest.Bold = True
```

## Описание Range объекта

**Range** –диапазон - непрерывная область текста в документе Word. Range однозначно определяется позициями начального и конечного символов заданной части текста. Range не зависит от области выделения текста. Диапазонов может быть много, а область выделения – одна.

### Использование метода Range

1. Set rngTest = ActiveDocument.Range (Start:=0, End:=10)  
rngTest.Bold = True
2. Set myRange = ActiveDocument.Content  
myRange.InsertAfter Text:= "New Text"
3. Dim rngTest As Range  
Dim wrdDoc As Document  
Set wrdDoc = ActiveDocument  
Set rngTest = wrdDoc.Range \_  
(Start:=wrdDoc.Paragraphs(2).Range.Start, \_  
End:=wrdDoc.Paragraphs(4).Range.End)  
rngTest.Select
4. Set myRange = ActiveDocument.Paragraphs(2).Range  
myRange.Select

### Использование свойства Range

1. Dim rngTest As Range  
Set rngTest = ActiveDocument.Paragraphs(1).Range
2. Set myRange = ActiveDocument.Paragraphs(1).Range  
With myRange  
.Bold = True  
.ParagraphFormat.Alignment = wdAlignParagraphCenter  
.Font.Name = "Arial"  
End With

## Переопределение Range

- Изменение начала (конца) Range  
Set rngCurrent = ActiveDocument.Paragraphs(2).Range  
rngCurrent.MoveEnd Unit:=wdCharacter, Count:=-1  
rngCurrent.InsertAfter " new text."
- Использование Collapse метода объекта Range  
Dim rngTest As Range  
Set rngTest = ActiveDocument.Paragraphs(1).Range  
With rngTest  
.Collapse Direction:=wdCollapseEnd  
.InsertAfter Text:="new text"  
.InsertParagraphAfter  
.Style = wdStyleHeading3  
End With

## Работа с выделенной областью (Selection)

### Выделенная область

```
Msgbox Selection.Text  
Selection.Homekey Unit:=wdStory  
Selection.MoveRight Unit:=wdWord, Count:=2, Extend:=wdExtend  
Selection.Font.Bold = wdToggle
```

То же, но используя объект Range:

```
Set rngTest = ActiveDocument.Range _
    (Start:=0, End:=ActiveDocument.Words(2).End)
rngTest.Font.Bold = True
```

#### Использование свойства Range объекта Selection

```
Dim rngTest As Range
Set rngTest = Selection.Range
rngTest.CheckSpelling
```

#### **Использование закладок (Bookmarks)**

определение, существует ли закладка и вставка текста на место закладки

```
If ActiveDocument.Bookmarks.Exists("CustName") = True Then
    ActiveDocument.Bookmarks("CustName").Range.InsertAfter "John"
End If
```

### ***7.4. Работа с текстом***

#### **Вставка и формат текста**

##### Вставка текста

1. Selection.InsertAfter "Thank you for calling."
2. Set rngTest = ActiveDocument.Paragraphs(2).Range  
rngTest.InsertAfter "Dear Sir:"  
rngTest.InsertParagraphAfter
3. Set rngTest = ActiveDocument.Paragraphs(2).Range  
rngTest.Text "Thank you for visiting the Post Office."

##### Форматирование текста

1. With rngTest  
With .Font  
.Bold = True  
.Name = "Arial"  
End With  
.ParagraphFormat.LeftIndent = InchesToPoints(0.5)  
End With
2. With ActiveDocument.Paragraphs(1)  
.Style = wdStyleHeading1  
.Alignment = wdAlignParagraphLeft  
End With
3. Sub AddBulletedList()  
' эта процедура добавляет маркированный текст в конец документа  
Dim rngCurrent As Range  
Set rngCurrent = ActiveDocument.Content  
rngCurrent.InsertParagraphAfter  
With rngCurrent  
.Collapse wdCollapseEnd  
.InsertAfter "Project 1"  
.InsertParagraphAfter  
.InsertAfter "Project 2"  
If .ListFormat.ListType = wdListNoNumbering Then  
rngCurrent.ListFormat.ApplyBulletDefault  
End If  
End With  
End Sub

## **Вставка автотекста**

```
Dim rngTest As Range
Dim tplCurrent As Template
Set rngTest = ActiveDocument.Content
Set tplCurrent = ActiveDocument.AttachedTemplate
rngTest.Collapse Direction:=wdCollapseEnd
tplCurrent.AutoTextEntries("Legal").Insert _
    Where:=rngTest
```

## **Поиск текста**

- **ПОИСК В ВЫДЕЛЕННОЙ ОБЛАСТИ**  
Selection.Find.Execute findtext:="company"
- **поиск в Range области**  
Dim rngDoc As Range  
Set rngDoc = ActiveDocument.Content  
rngDoc.Find.Execute FindText:="Monday"  
If rngDoc.Find.Found = True Then rngDoc.Bold = True
- **поиск в Range области с использованием With...EndWith**  
Set rngDoc = ActiveDocument.Content  
With rngDoc.Find  
    .ClearFormatting  
    .Text = "company"  
    .Forward = True  
    .Execute  
End With
- **поиск и замена в Range области**  
Set myRange = ActiveDocument.Content  
With myRange.Find  
    .ClearFormatting  
    .Text = "company"  
    .Execute replacewith:="Microsoft", replace:=wdReplaceAll  
End With
- **использование Do...Loop**  
Dim rngTest As Range  
Set rngTest = ActiveDocument.Content  
With rngTest.Find  
    .ClearFormatting  
    .Style = wdStyleHeading3  
    Do While .Execute(FindText:="", Forward:=True, Format:=True) = True  
        With rngTest  
            .StartOf Unit:=wdParagraph, Extend:=wdMove  
            .InsertAfter "Tip: "  
            .Move Unit:=wdParagraph, Count:=1  
        End With  
    Loop  
End With

## **Вставка полей и таблиц**

- **Вставка поля**  
Fields – коллекция объектов Field  
ActiveDocument.Fields.Add Range:=Selection.Range, \_  
    Type:=wdFieldDate, PreserveFormatting:=True

## ○ Вставка таблицы

```
Sub CreateTable()  
    Dim tabCurrent As Table  
    Dim rngCurrent As Range  
    Dim intRows As Integer  
    Dim intCols As Integer  
    Dim intBound As Integer  
    intBound = InputBox("Размер таблицы?", "Таблица умножения")  
    Set rngCurrent = ActiveDocument.Content  
    ' создание таблицы  
    Set tabCurrent = ActiveDocument.Tables.Add(Range:=rngCurrent, numRows:=intBound,  
numcolumns:=intBound)  
    ' заполнение таблицы  
    For intRows = 1 To intBound  
        For intCols = 1 To intBound  
            tabCurrent.Cell(intRows, intCols).Range = Str(intRows * intCols)  
        Next intCols  
    Next intRows  
    ' форматирование таблицы  
    tabCurrent.Cell(1, 1).Range = "*"   
    tabCurrent.AutoFormat (wdTableFormatGrid2)  
End Sub
```

## **7.5. Практическое занятие 6: Работа с Microsoft Word**

### **Задание 6.1 Автоматизация Word**

#### ➤ Создание нового документа

1. В папке \Labs\Lab07, откройте рабочую книгу Microsoft Excel с именем Stock.xls.

Вы используете данные из этой рабочей книги для создания нового документа Word, содержащего отчет об акциях для акционеров.

2. В редакторе Visual Basic создайте новый модуль и новую процедуру типа **Sub** с именем **CreateDocument**.
3. Используйте функцию **CreateObject** для создания нового экземпляра объекта Word.
4. Используйте метод **Add** коллекции Documents для создания нового документа.

#### ➤ Добавление содержимого к новому документу

1. Напишите код, который добавляет вводные слова к документу, например, «Биржевые цены акций для акционеров за текущий месяц», и пустой абзац в начале документа.
2. Опишите переменную типа **Chart**.
3. Напишите код, который использует переменную типа **Chart** для:
  - a) Копирования диаграммы на листе Sheet 1 в буфер обмена.
  - b) Вставки ее в конец документа Word.
4. Сделайте документ видимым.

Возможное решение:

```

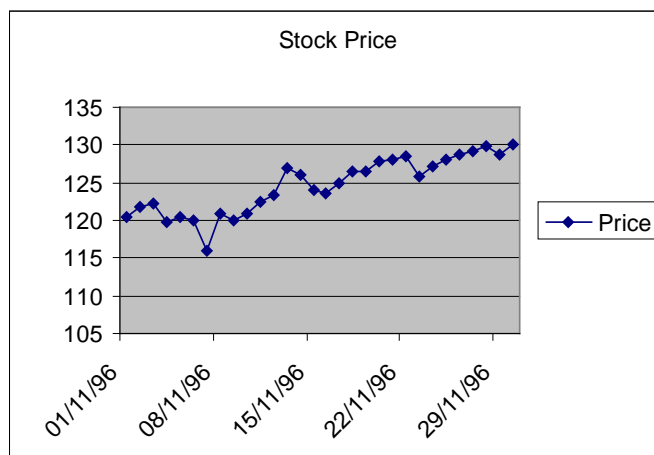
Sub CreateDocument()
    ' автоматизация Microsoft Word для создания нового
    ' документа на базе информации из таблицы.
Dim wrdApp As Word.Application
Dim wrdDoc As Word.Document, rngCurrent As Word.Range
Dim strIntro As String, chtStock As Chart
    ' создание нового экземпляра Word
Set wrdApp = CreateObject("Word.Application")
    ' добавить документ
Set wrdDoc = wrdApp.Documents.Add
    ' добавление содержимого к новому документа
strIntro = "Биржевые цены акций за текущий месяц" & vbCrLf
    ' use Word's объекта Range для создания некоторого теста
Set rngCurrent = wrdDoc.Content
rngCurrent.InsertParagraphAfter
rngCurrent.InsertAfter (strIntro)
rngCurrent.InsertParagraphAfter
rngCurrent.Collapse wdCollapseEnd
    ' скопировать диаграмму из excel
Set chtStock = Worksheets("sheet1").ChartObjects(1).Chart
chtStock.ChartArea.Copy
    rngCurrent.Paste ' вставить диаграмму в документ Word
    wrdApp.Visible = True ' сделать word видимым
End Sub

```

➤ Протестируйте процедуру

Ваша процедура должна создавать документ следующего вида:

Биржевые цены акций за текущий месяц



### Задание 6.2 Создание отчета

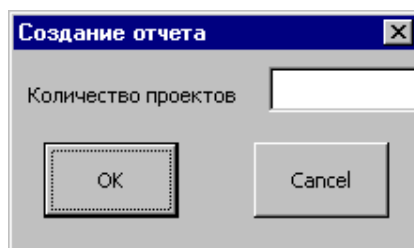
В этом задании вы создадите шаблон, который используется для создания отчета о проектах.

➤ Создание нового шаблона

1. Откройте Microsoft Word.
2. Создайте новый документ.
3. Сохраните документ в виде шаблона.

➤ Создание формы

1. В редакторе Visual Basic создайте новую форму, которая предназначена для ввода пользователем количества проектов, включаемых в новый отчет:



2. Напишите процедуру обработки события **Click** кнопки **OK**, в которой при вводе числового значения количества проектов вызывается процедура `BuildStatusReport` с аргументом количество проектов (процедура `BuildStatusReport`, которую вы создадите дальше, будет создавать основу для отчета). После ввода значения удалите (выгрузите) форму.

Возможное решение:

```
Private Sub cmdOK_Click()  
    If IsNumeric(txtNumProjects) Then  
        BuildStatusReport (CInt(txtNumProjects))  
    Else  
        MsgBox "Пожалуйста, введите числовое значение проектов"  
    End If  
    Unload Me  
End Sub
```

3. В событии **Click** для кнопки **Cancel** выгрузите форму.

➤ Реализация процедуры `BuildStatusReport`

1. Создайте новую процедуру с именем `BuildStatusReport` и одним аргументом типа `Integer`, которая создает оболочку для отчета.
2. В этой процедуре используйте цикл, создающий заголовок отчета для каждого проекта.
3. Для каждого заголовка проекта:
  - a. Вставьте новый текст заголовка для идентификации проекта.
  - b. Отформатируйте заголовок в виде стиля Заголовок 1 (Heading 1).
  - c. Вставьте список с маркерами для отчетных данных в каждом проекте.

Возможное решение:

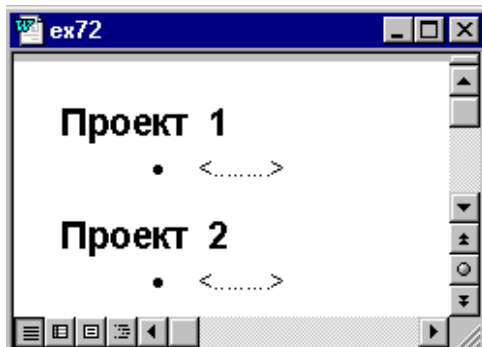
```
Sub BuildStatusReport(intNumberOfProjects As Integer)  
    Dim intProject As Integer  
    Dim rngCurrent As Range  
    For intProject = 1 To intNumberOfProjects  
        Set rngCurrent = ActiveDocument.Content  
        With rngCurrent  
            .Collapse wdCollapseEnd  
            .InsertAfter "Проект " & Str(intProject)  
            .Style = ActiveDocument.Styles(wdStyleHeading1)  
            .InsertParagraphAfter  
            .Collapse wdCollapseEnd  
            .InsertAfter "<Здесь информация о проекте>"  
            .ListFormat.ApplyBulletDefault  
        End With  
    Next intProject  
End Sub
```

```

.Paragraphs.Indent
.InsertParagraphAfter
.Collapse wdCollapseEnd
.ListFormat.ApplyBulletDefault
End With
Next intProject
End Sub

```

Полный отчет для проекта может выглядеть следующим образом.



➤ Реализация построителя отчетов

1. Создайте модуль, в нем процедуру, которая выводит форму (UserForm1.Show).
2. Реализуйте интерфейс управления вызовом этой процедуры.

**Задание 6.3 Создание пользовательского документа**

В этом задании вы создадите процедуру, которая выводит форму. С помощью этой формы пользователь может выбрать значения **авто текста** для создания нового документа.

Шаблон Ex3Start.Dot в папке \Labs\Lab07 содержит **закладки** (bookmarks) с именами Name1, Name2, и LetterText. Шаблон также включает данные Автотекста (AutoText) с именами Intro, Catalog, Discount, Sampler, и Thanks.

Вы добавите код к шаблону для вывода формы с подсказкой пользователю для ввода имени и адреса и значений из авто текста для включения в документ. Затем информация, введенная в форме, используется для построения пользовательского письма.

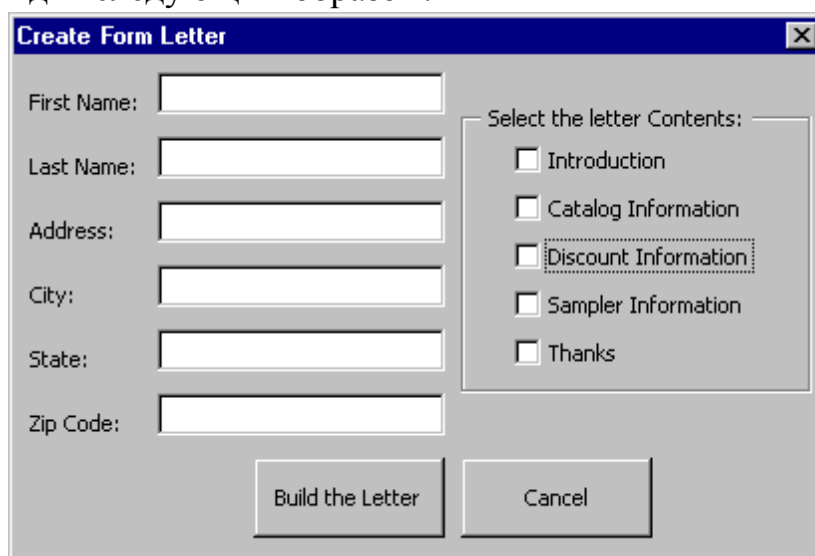
➤ Открытие шаблона

1. Откройте шаблон Ex3Start.dot в папке \Labs\Lab07 (для открытия шаблона сделайте правый щелчок в окне Windows Explorer, затем щелкните **Открыть**, если вы сделаете двойной щелчок, Word создаст новый документ, основанный на шаблоне, а не откроет шаблон)
2. Для просмотра закладок включите эту функциональность: **параметры Word → Дополнительно → в разделе Показывать содержимое документа → показывать Закладки → ОК** (вы увидите положение закладок, уже расположенных в шаблоне).
3. Для просмотра содержимого **Авто текста** в шаблоне, выберите на вкладке **Вставка → в группе Текст → раскройте кнопку Экспресс-блоки → нажмите Организатор стандартных блоков**. Просмотрите коллекцию **Автотекст** шаблона Ex3Start.dot.



➤ Открытие формы

1. В редакторе Visual Basic откройте форму, включенную в проект. Форма выглядит следующим образом.



2. Для события **Click** кнопки **Cancel** добавьте код выгружающий форму.
3. Для события **Click** кнопки **Build the Letter**:
  - a. Вставьте имя и фамилию, введенные в форме, в закладку Name1.
  - b. Вставьте адрес в закладку Address.
  - c. Вставьте имя в закладку Name2.
  - d. Определите, какие значения из содержимого Автотекст (AutoText) необходимо использовать и вставьте их в закладку LetterText.
  - e. Добавьте содержимое Автотекст (AutoText) в следующем порядке: Intro, Catalog, Discount, Sampler, и Thanks.
  - f. Выгрузите форму.

Возможное решение:

```
Private Sub cmdBuild_Click()  
    Dim strAddress As String, rngCurrent As Range  
    ActiveDocument.Bookmarks(«Name1»).Range.InsertAfter _  
        (txtFirstName.Text & " " & txtLastName.Text)  
    strAddress = txtAddress.Text & vbCrLf & txtCity.Text & ", " & _  
        txtState.Text & " " & txtZipCode.Text & vbCrLf  
    ActiveDocument.Bookmarks(«Address»).Range.InsertAfter strAddress  
    ActiveDocument.Bookmarks(«Name2»).Range.InsertAfter txtFirstName.Text  
    Set rngCurrent = ActiveDocument.Bookmarks(«LetterText»).Range  
    rngCurrent.Select  
    With ActiveDocument.AttachedTemplate  
        If chkIntro = True Then  
            rngCurrent.InsertAfter .AutoTextEntries(«Intro»)  
        End If  
        If chkCatalog = True Then  
            rngCurrent.InsertAfter .AutoTextEntries(«Catalog»)  
        End If  
        If chkDiscount = True Then  
            rngCurrent.InsertAfter .AutoTextEntries(«Discount»)  
        End If  
        If chkSampler = True Then
```

```
rngCurrent.InsertAfter .AutoTextEntries(«Sampler»)
    End If
    If chkThanks = True Then
rngCurrent.InsertAfter .AutoTextEntries(«Thanks»)
    End If
End With
```

Unload Me

End Sub

➤ Вывод формы

Добавьте код к событию **New** шаблона. Форма должна появляться при создании нового документа на основе шаблона.

➤ Протестируйте ваш шаблон

1. Создайте документ, основанный на шаблоне. Для создания документа, основанного на шаблоне, сделайте двойной щелчок по шаблону в Проводнике (Windows Explorer).
2. Заполните форму и затем щелкните ОК.

Должен быть создан новый документ с соответствующим текстом.

## 8. Работа с базами данных

### 8.1. Обзор технологии ADO

Основным методом доступа к разделяемым файлам и базам данных клиент/сервер является технология **ActiveX Data Objects (ADO)** и OLE DB, пришедшие на смену прикладному интерфейсу программирования **Open Database Connectivity (ODBC)**.

ADO поддерживает ключевые возможности для построения клиент/серверных и Web-приложений, а также обеспечивает функции Remote Data Service (RDS), посредством которого можно перемещать данные с сервера в клиентское приложение или на Web-страницу, манипулировать данными «на стороне клиента» и возвращать обновленные данные серверу.

**ADO (ActiveX Data Objects)** - набор программных объектов, построенных по технологии ActiveX (COM) – универсальный объектно-ориентированный программный интерфейс для сетевых и локальных баз данных, разработанный фирмой Microsoft в рамках относительно новой технологии доступа к данным OLE DB и призванный заменить интерфейс DAO в практике проектирования приложений.

**Основное достоинство** ADO по сравнению с DAO состоит в универсальности подключения к источникам данных различного типа.

Вобъектную модель ADO включены три главных объекта:

- **Connection**
- **Command**
- **Recordset**

Эти объекты являются высокоуровневым интерфейсом для объектов OLE DB, которые обеспечивают непосредственный доступ к данным.

**Connection** — позволяет установить соединение с источником данных и управлять им. Объект Connection обеспечивает соединение с одним источником данных и управляет в рамках этого соединения транзакциями. Соединение может быть установлено как с текущей базой данных Access, так и с внешним источником любого специфицированного типа.

Все ошибки, которые возникают в ходе работы соединения, помещаются в сопутствующую коллекцию **Errors**.

**Command** — представляет собой команду, при помощи которой производится выполнение определенной операции на источнике данных (выполнение запроса, хранимой процедуры, создание или изменение объекта, изменение данных и т. п.). Если источник данных SQL-совместимый, то объект Command, скорее всего, будет представлять команду SQL. Объекту Command сопутствует коллекция **Parameters** — параметры, которые передаются запросу или хранимой процедуре.

**Recordset** — является набором записей, полученных с источника или сгенерированных другим способом. Этому объекту сопутствует коллекция

**Fields**, представляющая информацию о столбцах в этом наборе записей (имя, тип, размерность данных и т. п.), а также сами данные.

Для каждого из этих трех объектов предусмотрена также коллекция **Properties**, которая определяет свойства соединения, команды или набора записей соответственно.

Все объекты явно создавать необязательно. Так, например, при создании объекта **Recordset** можно в автоматическом режиме создать объект **Connection**.

#### Объект Connection

Объект **Connection** — основной объект ADO верхнего уровня. Именно с него начинается подключение к источнику данных, после чего можно использовать связанные с соединением объекты **Command** или **Recordset**.

Фактически все, что нужно для открытия соединения с базой данных:

- создать объект **Connection**,
- настроить для него свойство **ConnectionString**
- и вызвать метод **Open()**.

Основные свойства и методы объекта:

**Provider** — это свойство позволяет определить драйвер, который будет использован для подключения к базе данных. Его можно определить внутри значения **ConnectionString**, но можно для этой цели использовать и отдельное свойство.

Значения свойств **Provider** для подключения к разным источникам данных могут выглядеть так:

- "Microsoft.Jet.OLEDB.4.0" — для подключений к файлам Access;
- "SQLOLEDB.1" — для подключений к SQL Server (как в примере);
- "MSDAORA.1" — для подключений к серверу Oracle;
- "ADsDSOObject" — для подключения к базе данных службы каталогов Windows.

**ConnectionString** — это главное свойство объекта **Connection**. Оно определяет параметры подключения к источнику.

**Open()** — этот метод позволяет открыть соединение с базой данных. Строку подключения можно не настраивать отдельно как свойство объекта **Connection**, а просто передавать ее этому методу как параметр.

**Close()** — позволяет закрыть соединение (объект соединения при этом из памяти не удаляется). Чтобы полностью избавиться от этого объекта, можно использовать код:

```
cn.Close  
Set cn = Nothing
```

или:

```
Set cn = Nothing
```

разрыв соединения произойдет автоматически.

Таким образом, для **установления соединения** необходимо описать новую объектную переменную типа **Connection** и затем определить для нее источник данных.

В общем случае источник определяется названием провайдера и спецификацией файла с данными.

Строка с информацией о провайдере должна быть присвоена свойству **Provider** объекта **Connection**, а строка с путем доступа к файлу источника - свойству **ConnectionString**. После этого должен быть вызван метод **Open** объекта **Connection**.

Например, чтобы подключиться к базе данных Nwind, можно использовать код типа:

```
Dim cn As New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Nwind.mdb"
cn.Open
```

Для объекта **Connection** также предусмотрено множество других свойств и методов.

Единственное свойство, которое обязательно нужно знать, — это свойство **Errors**, которое возвращает коллекцию объектов **Error** — ошибок.

Ошибки при установке или работе соединения: неверно введен пароль или имя пользователя, у пользователя недостаточно прав на подключение, невозможно обратиться к компьютеру по сети и т. п.

Рекомендуется реализовывать в программе обработку ошибок.

Пример. Реализация обработчика ошибок

```
Dim cn As ADODB.Connection
Set cn = CreateObject("ADODB.Connection")
cn.Provider = "SQLOLEDB"
cn.ConnectionString = "User ID=SA;Password=password;" _
& "Data Source = LONDON1;Initial Catalog = Northwind"
On Error GoTo CnErrorHandler
cn.Open
Exit Sub
CnErrorHandler:
For Each ADOErr In cn.Errors
Debug.Print ADOErr.Number
Debug.Print ADOErr.Description
Next
```

На практике при возникновении ошибки пользователю предлагается ее исправить и еще раз произвести подключение.

Для получения информации о том, почему возникла ошибка, используется специальный объект **ADOError** (при возникновении ошибки он создается автоматически).

Основные свойства объекта **ADOError**.

**Description** — описание ошибки. Обычно наиболее важная информация содержится именно в описании.

**Number** — номер ошибки. По номеру удобно производить поиск в базе знаний Microsoft ([www.microsoft.com/support](http://www.microsoft.com/support)) и в Интернете.

**Source** — источник ошибки. Эта информация полезна только в том случае, если в коллекции **Errors** могут оказаться ошибки из разных источников.

**SQLState** и **NativeError** — информация о возникшей ошибке, которая пришла с SQL-совместимого источника данных.

## Объект **Recordset** и коллекция **Fields**

После установления соединения с источником данных можно выполнять манипулирование данными — добавлять и удалять записи таблиц, имеющихся в источнике, изменять значения их полей.

Объект **Recordset** (*Set of Records*, т. е. набор записей) предназначен для доступа к наборам записей в источнике данных.

В ADO наборы записей объектов **Recordset** называются *курсорам*.

Можно представить объект **Recordset** как таблицу (аналогичную таблицам в Excel), которая находится в оперативной памяти компьютера.

Однако у **Recordset** есть принципиальные отличия от таблиц Excel:

- Excel не следит за "строгостью" таблиц. **Recordset** — это "строгая" таблица. В ней четко определены столбцы и строки и разрывов она не допускает (хотя какие-то значения на пересечении строк и столбцов вполне могут быть пустыми);
- в таблице Excel в одном столбце без проблем можно использовать самые разные значения — числовые, даты и времени, строковые, формулы и т. п. В **Recordset** для каждого столбца определяется тип данных и значения в этом столбце должны соответствовать этому типу данных.
- **Recordset** обычно создается на основе данных, полученных с источника (но может быть создан и заполнен вручную), в которых предусмотрены столбцы (**Fields**) и строки (**Rows**).

Для создания объекта **Recordset** ADO необходимо описать новую объектную переменную типа **Recordset** и вызвать ее метод **Open**, указав в качестве его параметра имя таблицы или хранимого запроса в источнике, или инструкцию SQL на выборку данных из таблицы или таблиц.

Создание объекта **Recordset** и заполнение его данными с источника в самом простом варианте выглядит так (подразумевается, что мы открыли при помощи объекта **cn** соединение с учебной базой данных **Nwind**):

```
Dim rs As New ADODB.Recordset  
rs.Open "Employees", cn
```

В нашем примере мы открыли таблицу **Customers** целиком. Однако это не единственный (и не лучший) способ извлечения данных с источника. Для метода **Open()** рекомендуется использовать запрос на языке SQL.

Например, в нашем случае можно было бы использовать такой код:

```
rs.Open "SELECT Employees.FirstName, Employees.Title, Orders.OrderDate, Orders.ShipName  
FROM Employees INNER JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID", cn
```

Преимущества использования запроса следующие:

- есть возможность указать фильтр **Where** (условие обязательно заключать в одинарные кавычки) и скачать в **Recordset** не все записи, а только удовлетворяющие вашему условию;
- есть возможность точно так же ограничить число возвращаемых столбцов (снова сокращение объема передаваемых данных и уменьшение расхода памяти);
- есть возможность использовать функции SQL, сортировку на источнике данных и множество полезных дополнительных возможностей.

Очень часто в реальных приложениях текст запроса "склеивается" из кусочков, которые поступают из разных мест.

Например, пользователь выбрал в раскрывающемся списке имя заказчика, и для события Change этого списка тут же сработала процедура, которая выполнила запрос на SQL Server, получив данные только для этого заказчика, и присвоила полученные значения другим элементам управления.

Соответствующая строка кода может выглядеть так:

```
rs.Open "select * from dbo.customers Where CompanyName = " & "" _  
& ComboBox1.Value & "" , cn
```

Набор символов "" — это одинарная кавычка внутри двух двойных.

Такая конструкция нужна, чтобы текст запроса мог выглядеть, например, так:

```
select * from dbo.customers Where CompanyName = 'Alfreds Futterkiste'
```

Первый параметр метода **Open()** может быть как именем таблицы, так и командой SQL. Поскольку драйвер OLE DB не знает, чем может быть передаваемый текст, он взаимодействует с сервером баз данных, чтобы определить это.

На практике такое выяснение может сильно тормозить работу приложения, поэтому имеет смысл перед открытием **Recordset** явно указать тип передаваемых параметров. Это делается при помощи параметра **Options**, который передается этому методу.

Наиболее часто используемые значения такие:

**adCmdText** — передается команда SQL;

**adCmdTable** — передается имя таблицы (равносильно указанию сгенерировать команду SQL, которая вернет все записи из таблицы);

**adCmdTableDirect** — также передается имя таблицы для того, чтобы получить все ее записи напрямую (без выполнения SQL-запроса), если источник поддерживает такую операцию;

**adCmdStoredProc** — передается имя хранимой процедуры для ее выполнения, а то, что она вернет, используется для заполнения Recordset.

С практической точки зрения важно запомнить следующее. Если вам нужно обеспечить себе возможность перемещения по Recordset в любом направлении и изменять в нем записи, код на открытие Recordset должен быть таким:

```
Dim rs As New ADODB.Recordset  
rs.CursorType = adOpenStatic  
rs.LockType = adLockOptimistic  
rs.Open ... 'Пишем, что именно мы открываем
```

### Свойства и методы - Перемещение по *Recordset*

После того как объект Recordset создан, нам необходимо выполнять с ним различные операции.

Самое простое действие, с которого мы начнем, — перемещение по объекту Recordset.

В Recordset всегда имеется ограниченное количество записей (столько, сколько мы получили с источника). Изначально курсор устанавливается на

первую запись в Recordset (убедиться в этом можно при помощи свойства AbsolutePosition). Однако если мы выполним команду **MovePrevious()** (но только один раз), ошибки не произойдет, а если мы попробуем выполнить команду **MovePrevious()** второй раз, то возникнет ошибка. AbsolutePosition вернет загадочное значение -2. Связано это с тем, что в Recordset перед первой записью, полученной с источника, помещается специальная запись **BOF** (от англ. Begin Of File, хотя никаких файлов, конечно же, нет).

Проверить, находимся ли мы на этой специальной записи, можно при помощи **свойства BOF**.

Например, такой код (если он выполнен сразу после открытия Recordset):

```
Debug.Print rs.BOF
rs.MovePrevious
Debug.Print rs.BOF
```

вернет нам вначале False, а затем True.

Точно так же после последней записи в **Recordset** находится специальная запись **EOF** (от End Of File). Проверить, не находится ли курсор на ней, можно при помощи аналогичного одноименного свойства **EOF**.

Иногда бывает так, что сразу после открытия Recordset и BOF, и EOF одновременно возвращают True. Объяснение такой ситуации очень простое — в Recordset с источника по каким-то причинам не вернулось ни одной записи. Рекомендуется во избежание неожиданностей предусматривать сразу после открытия Recordset проверку на наличие в нем записей.

После того, как мы определились с текущей позицией в Recordset, необходимо разобраться с тем, как можно по нему перемещаться.

Проще всего это делать при помощи методов с префиксом **Move...**

**Move()** — этот метод принимает два параметра: *NumRecords* — на сколько записей необходимо переместиться (это число может быть и отрицательным, что означает переместиться назад) и второй параметр (необязательный) — имя закладки, с которой нужно начать перемещение. Можно использовать три встроенные закладки: для текущей, первой и последней записи. Если имя закладки не указано, то перемещение начинается с текущей позиции.

**MoveFirst()**, **MoveLast()**, **MoveNext()** и **MovePrevious()** — назначения этих методов понятны из названий: перемещение на первую, последнюю, следующую и предыдущую запись соответственно.

Необходимо отметить, что перемещение назад (при помощи **MovePrevious()** или **Move()** с отрицательным значением) для курсора, открытого как **adOpenForwardOnly**, может привести к совершенно непредсказуемому результату (в зависимости от источника данных) — от ошибки до перехода на случайную запись.

Чаще всего для прохода по всем записям используется такой алгоритм:

```
rs.MoveFirst
Do Until rs.EOF
'Что-то делаем с каждой записью
'Например, получаем значение нужного поля
rs.MoveNext
Loop
```



Если необходимо напрямую перепрыгнуть на нужную запись, можно использовать методы **Find()** и **Seek()**.

**Find()** — метод предназначен для поиска по значению одного столбца. Он принимает в качестве параметра критерий поиска, насколько нужно отступить от исходной позиции, направление поиска и откуда нужно начать поиск. Очень удобно, что при определении критерия поиска можно использовать оператор **Like** с подстановочными символами. При обнаружении нужной записи метод **Find()** переставляет курсор на найденную запись, если же запись не обнаружена, то курсор устанавливается на EOF (или BOF, если поиск был в обратном порядке).

Например, чтобы найти все немецкие фирмы в нашем **Recordset** для таблицы Customers, можно использовать код вида:

```
rs.Find "country = 'Germany'"
Do While Not rs.EOF
Debug.Print "Название фирмы: " & rs.Fields("CompanyName")
mark = rs.Bookmark
rs.Find "country = 'Germany'", 1, adSearchForward, mark
Loop
```

В этом примере используются еще незнакомые нам объекты **Fields** и **Bookmark**, но их назначение понятно: объект **Field** нужен, чтобы вывести название фирмы, а объект **Bookmark** — чтобы продолжить поиск, начиная со следующей записи по отношению к последней найденной.

**Seek()** — отличается от метода **Find()** тем, что он ищет значение по индексу (объект **Index** для **Recordset** создается либо программным способом, либо автоматически, если на таблицу, на основе которой был создан **Recordset**, было наложено ограничение первичного ключа (**Primary Key**)). Этот метод работает только для серверных курсоров с типом команды **TableDirect**, и поэтому к использованию не рекомендуется.

## 8.2. Коллекция **Fields** и объекты **Field**

Главное содержание **Recordset** — это то, что лежит в ячейках на пересечении **строка** (в **Recordset** они называются **записями** (**records**)) и представлены объектами **Record**) и столбцов.

В **Recordset** **столбцы** называются **полями** и представляются объектами **Field**, которые сведены в коллекцию **Fields**.

Объекты **Record** используются нечасто, поскольку имен у них нет, а переходить между записями проще при помощи свойств и методов самого объекта **Recordset** — **AbsolutePosition**, **Find()**, **Move()** и т. п.

Коллекция же **Fields** и объекты **Field** используются практически в каждой программе.

У коллекции **Fields** все **свойства** стандартные, как у каждого объекта **Collection**.

**Count** — возвращает, сколько всего столбцов в **Recordset**.

**Item** — позволяет вернуть нужный столбец (объект **Field**) по имени или номеру. Поскольку это свойство является свойством по умолчанию, то можно использовать код, как в нашем примере:

```
rs.Fields("CompanyName")
```

Есть еще один вариант синтаксиса для обращения к этому свойству:

```
rs!CompanyName
```

**Методы** у этой коллекции есть как стандартные, так и специфические.

**Append()** — добавляет новый столбец в Recordset. **Delete()** — удаляет столбец. Обе команды разрешено выполнять только на закрытом Recordset (пока не был вызван метод **Open()** или не установлено свойство ActiveConnection).

**Update()** — сохраняет изменения, внесенные в Recordset. Метод **CancelUpdate()** — отменяет изменения, внесенные в Recordset.

**Refresh()** — загадочный метод, который ничего не делает (о чем честно написано в документации). Обновить структуру Recordset данными с источника можно только методами самого объекта Recordset.

**Resync()** — работает только для коллекции **Fields** объекта Record (не Recordset), обновляя значения в строке.

Намного больше **интересных свойств** у объекта **Field**.

**ActualSize** — реальный размер данных для текущей записи, **DefinedSize** — номинальный размер данных для столбца (в байтах), в соответствии с полученной с источника информацией.

**Attributes** — определяет битовую маску для атрибутов столбца (допускает ли пустые значения, можно ли использовать отрицательные значения, можно ли обновлять, используется ли тип данных фиксированной длины и т. п.)

**Name** — просто строковое имя столбца. Для столбцов, полученных с источника, это свойство доступно только для чтения.

**NumericScale** и **Precision** — значения, которые определяют соответственно допустимое количество знаков после запятой и общее максимальное количество цифр, которое можно использовать для представления значения.

**Value** — самое важное свойство объекта Field. Определяет значение, которое находится в столбце (если мы пришли через коллекцию Fields объекта Record, то значение этой записи Record; если через Fields объекта Recordset — текущей записи). Доступно и для чтения, и для записи (в зависимости от типа указателя).

ADO позволяет работать с большими двоичными данными (изображения, документы, архивы), что очень удобно.

Свойство **OriginalValue** возвращает значение, которое было в этом столбце до начала изменений, **UnderlyingValue** — значение, которое находится на источнике данных (пока мы работали с Recordset, оно могло быть изменено другим пользователем, и поэтому **OriginalValue** и **UnderlyingValue** могут не совпадать).

Свойство **Value** — это свойство объекта **Field** по умолчанию (т. е. то свойство, значение которого будет возвращаться, если не указывать, к какому свойству объекта мы обращаемся), поэтому следующие две строки равноценны:

```
Debug.Print rs.Fields("CompanyName")  
Debug.Print rs.Fields("CompanyName").Value
```

**Status** — значение этого свойства, отличное от `adFieldOK` (значение 0), означает, что поле было недавно программно добавлено в `Recordset`.

**Type** — определяет тип данных столбца в соответствии с приведенной в документации таблицей. Например, для типа данных `nvarchar` возвращается 202.

У объекта `Field` есть только два метода: **AppendChunk()** и **GetChunk()**. Оба эти метода используются только для работы с большими двоичными типами данных (изображениями, документами и т. п.), когда использовать обычными способами свойство `Value` не получается.

### 8.3. Сортировка и фильтрация данных

Данные в **Recordset** помещаются в том порядке, как они пришли из источника.

Если специальный порядок сортировки в запросе не указан, то данные возвращаются в соответствии с параметрами источника данных (например, на `SQL Server` они будут по умолчанию упорядочены по кластерному индексу, который по умолчанию создается для первичного ключа).

Можно произвести сортировку на сервере, указав в запросе выражение `ORDER BY`, а можно выполнить сортировку на клиенте при помощи свойства `Sort` объекта `Recordset`.

**Общее правило** выглядит так: если есть возможность, всегда нужно выполнять сортировку на сервере.

Сервер намного лучше оптимизирован для выполнения подобных операций, на нем обычно больше оперативной памяти и процессорных ресурсов.

На клиенте сортировку выполнять следует только тогда, когда это невозможно сделать на сервере (например, вы получаете данные от хранимой процедуры, в тексте которой сортировка не предусмотрена). Однако, в принципе, сортировка в `Recordset` менее ресурсоемка, чем могла бы быть (данные физически не перемещаются, только создается новый индекс для поля, по которому производится сортировка), поэтому ее вполне можно использовать в программах даже для большого количества данных.

Применение свойства **Sort** связано с двумя серьезными ограничениями:

- его можно использовать только тогда, когда курсор открыт на клиенте (т. е. для свойства **CursorLocation** должно быть установлено значение **adUseClient**, по умолчанию он открывается на сервере);
- это свойство нельзя использовать с некоторыми драйверами, например, нельзя сортировать данные при подключении к `Excel`.

Применение этого свойства выглядит очень просто:

```
rs.Sort = "CompanyName"
```

При этом то, что передается этому свойству, должно быть названием столбца (т. е. именем объекта **Field**) в `Recordset`.

Можно передавать несколько названий столбцов и разные порядки сортировки:

```
rs.Sort = "Country ASC, CompanyName DESC"
```

Recordset вначале будет отсортирован по стране (Country), а потом — по имени компании (CompanyName) в убывающем порядке DESC (по умолчанию используется ASC — по возрастанию, поэтому в нашем примере это слово можно опустить).

Чтобы отменить сортировку (и вернуться к записям в том порядке, в котором они были возвращены с источника), достаточно присвоить этому свойству пустое значение:

```
rs.Sort = ""
```

В Recordset записи можно **фильтровать**.

Отфильтрованные записи остаются в Recordset, но являются невидимыми при выполнении операций перемещения и поиска, и курсор на отфильтрованных записях установить нельзя.

Для фильтрации используется свойство **Filter** объекта Recordset.

Оно может принимать три типа значений:

1. строковое значение, по синтаксису аналогичное передаваемому методу Find():

```
rs.Filter = "LastName = 'Smith' AND FirstName = 'John'"
```

Можно использовать оператор Like с подстановочными символами (только '\*' и '%'). Отличие от Find() заключается в том, что в Find() просто устанавливается курсор на первую найденную подходящую запись, а в Filter все неподходящие становятся невидимыми;

2. массив закладок;

3. несколько специальных значений — все конфликтующие записи, записи, ожидающие сохранения на источнике, и т. п.

Снять фильтрацию можно точно так же, как и сортировку:

```
rs.Filter = ""
```

#### **8.4. Объект Command и коллекция Parameters**

В самых простых случаях, когда можно получать и изменять данные напрямую в таблицы, можно обойтись объектом Recordset.

Однако во многих ситуациях возможностей этого объекта недостаточно. Как уже говорилось ранее, предпочтительнее производить любое внесение изменений на источник данных при помощи хранимых процедур. Часто существует потребность в создании временных таблиц и других объектов на сервере. Бизнес-логика многих приложений (начисление процентов, абонентской платы, формирование специальных отчетов с вычислениями и т. п.) также реализована в виде хранимых процедур, поэтому в реальных приложениях одним объектом Recordset не обойтись.

**Для выполнения команд SQL на сервере** (в том числе для запуска хранимых процедур, команд DDL для создания объектов, выполнения служебных операций типа резервного копирования, восстановления, изменения параметров работы) **необходимо использовать объект Command**.

**Создание** этого объекта производится очень просто:

```
Dim cmd As ADODB.Command  
Set cmd = CreateObject("ADODB.Command")
```

Следующее, что нужно сделать, — это **назначить объекту Command** объект подключения Connection.

Для этой цели предназначено свойство **Command.ActiveConnection**.

Ему можно передать готовый объект Connection, а можно сформировать этот объект неявно, используя в качестве значения свойства ActiveConnection строку подключения.

Рекомендуется всегда предавать готовый объект подключения:

во-первых, так для соединения можно настроить больше параметров, а во-вторых, если вы используете в приложении несколько объектов Command, можно использовать для каждого такого объекта одно-единственное подключение, что экономит ресурсы.

В нашем примере мы используем созданный нами ранее объект Connection:

```
cmd.ActiveConnection = cn
```

Следующая наша задача — **выбрать тип команды**.

В принципе, для многих источников можно этого не делать — модули ADO постараются сами выяснить у источника данных, что это за команда (храняемая процедура, SQL-запрос и т. п.), однако лучше всегда его определять: экономится время и системные ресурсы, уменьшается вероятность ошибок. Для выбора типа команды используется свойство **CommandType**. Значения, которые ему можно присвоить, аналогичны возможным значениям параметра Options метода Open() объекта Recordset.

Например, если мы передаем команду на выполнение хранимой процедуры, то присвоить соответствующее значение можно так:

```
cmd.CommandType = adCmdStoredProc
```

Следующее действие — **определить текст команды**, которая будет выполняться.

Делается это при помощи свойства **CommandText**.

Например, если хотим запустить на выполнение хранимую процедуру CustOrderHist, то соответствующий код может выглядеть так:

```
cmd.CommandText = "CustOrderHist"
```

Чаще всего хранимая процедура требует передачи ей одного или нескольких параметров. Делается это при помощи коллекции **Parameters** и объектов **Parameter**.

Для **определения параметров** можно использовать два способа:

- вначале создать объекты **Parameter** автоматически путем запроса к серверу (используется метод Refresh() коллекции Parameters), а затем присвоить им значения:

```
cmd.Parameters.Refresh  
cmd.Parameters(1) = "ALFKI"
```

- создать объекты **Parameter** вручную и вручную добавить их в коллекцию **Parameters**. Этот способ более экономичен (нет необходимости лишней раз обращаться на сервер), но требует предварительно знать точные свойства параметра и использовать код большего размера:

```
Dim Prm As ADODB.Parameter
```

```
Set Prm = cmd.CreateParameter("CustomerID", adVarChar, _  
adParamInput, 5, "ALFKI")  
cmd.Parameters.Append Prm
```

После определения параметров команду необходимо запустить на выполнение. Для этого используется метод **Execute()**. Самый простой способ его вызова выглядит так:

```
cmd.Execute
```

Этот метод принимает также три необязательных параметра, при помощи которых можно дополнительно определить параметры, тип вызываемой команды и т. п.

Некоторые хранимые процедуры и передаваемые команды не требуют возврата каких-либо значений (кроме кода ошибки), но это бывает редко.

Как же принять значение, возвращаемое выполняемой командой?

Если возвращаемое значение официально зарегистрировано как возвращаемый параметр (например, оно помечено ключевым словом **OUT** в определении хранимой процедуры), то это значение будет присвоено соответствующему параметру объекта **Command**, и до него можно будет добраться обычным способом — при помощи свойства **Value**.

Если же, как в нашем примере с **CustOrderHist**, возвращаемое значение просто сбрасывается в поток вывода (в нашем случае возвращается набор записей), то можно использовать два способа:

- первый способ — использовать то, что метод **Execute()** возвращает объект **Recordset**, заполненный полученными при помощи команды записями:

```
Dim rs2 As ADODB.Recordset  
Set rs2 = cmd.Execute()  
Debug.Print rs2.GetString
```

- второй способ — воспользоваться тем, что метод **Open()** объекта **Recordset** может принимать в качестве параметра объект **Command** (в этом случае объект **Connection** передавать этому методу уже нельзя):

```
Dim rs2 As ADODB.Recordset  
Set rs2 = CreateObject("ADODB.Recordset")  
rs2.Open cmd  
Debug.Print rs2.GetString
```

Другие свойства и методы объекта **Command**:

**CommandStream** — позволяет вместо прямого назначения текста команды (через свойство **CommandText**) принять значение из потока ввода (например, из текстового файла или другой программы).

**CommandTimeout** — позволяет указать, сколько времени в секундах ждать результата выполнения команды на источнике, прежде чем вернуть ошибку.

**Dialect** — это свойство позволяет указать особенности разбора (parsing) текста команды на провайдере.

**NamedParameters** — определяет, будут ли передаваться провайдеру имена параметров (**True**) или будет использоваться простая передача значений по порядку (**False**, по умолчанию).

**Prepared** — свойство, которое может влиять на производительность. Если установить его в True (по умолчанию False), то при первом выполнении команды провайдер создаст ее откомпилированную версию, которую и будет использовать при последующих выполнениях. Первый раз команда будет выполняться медленнее, чем обычно, зато в следующие разы — быстрее.

**State** — возвращает те же значения и используется в тех же целях, что и для объекта Recordset.

**Cancel()** — этот метод позволяет прекратить выполнение команды (когда выполнение затянулось), если такую возможность поддерживает провайдер.

### **8.5. Практическое занятие 7: Работа с базой данных**

На этом занятии Вы подключитесь к базе данных, создадите Query Table на основе объекта **RecordSet**, в котором хранится информация, полученная из базы данных.

#### **Задание 7.1. Загрузка данных**

В компании ведется учет товаров, которые имеются на складе, при помощи таблицы **Products** базы данных **Nwind**.

В этой таблице находятся следующие важные столбцы:

- *Product ID* — идентификатор товара;
- *ProductName* — наименование продукта;
- *UnitPrice* — стоимость продукта за единицу;
- *UnitsInStock (На складе)* — количество единиц этого товара на складе;
- *ReorderLevel (Минимальный запас)* — минимально допустимое количество единиц данного товара на складе. Если реальное количество единиц этого товара меньше, чем этот уровень, товар нужно срочно заказывать;
- *Discontinued (Поставки прекращены)* — флаг прекращения работы с товаром. Если в этом столбце стоит «истина», то это значит, что принято решение закупки этого товара больше не производить.

Все остальные столбцы для целей этой работы можно игнорировать.

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

#### Требуется

Создать приложение на основе Excel, которое:

- Производит вставку в лист Excel данные по всем строкам и всем вышеуказанным столбцам этой таблицы.
- Генерирует в Excel дополнительные столбцы следующего содержания:  
*Заказать товар, итук* — разница между столбцами **ReorderLevel (Минимальный запас)** и **UnitsInStock (На складе)**. В этот столбец помещается информация о количестве товара в штуках, которое нужно срочно заказать.

Эту информацию нужно генерировать только для тех записей, для которых значение в столбце **ReorderLevel (Минимальный запас)** больше, чем в

столбце **UnitsInStock** (**На складе**), и у которых значение столбца **Discontinued** (**Поставки прекращены**) установлено в **Ложь**.

*Стоимость заказа* — определяло бы стоимость такого пополнения склада для каждой строки в таблице. Стоимость заказа рассчитывается как произведение предыдущего столбца и столбца **UnitPrice** (**Цена**).

Эту информацию также нужно генерировать только для тех записей, для которых значение в столбце **ReorderLevel** (**Минимальный запас**) больше, чем в столбце **UnitsInStock** (**На складе**).

- Вставляло бы одной строкой ниже полученных записей из базы данных две итоговые строки:
  - **общая стоимость товаров на складе** — итоговая стоимость всех товаров, которые находятся на складе (как сумма произведений столбцов **На складе** и **Цена** для каждой строки);
  - **общая стоимость товаров к заказу** — итог по столбцу **Стоимость заказа**.

Общий вид получившегося приложения может быть таким, как представлено на рис. 8.1:

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

*Рис. 8.1 Первые строки листа с импортированными данными*

Итоговые строки могут выглядеть так, как показано на рис. 8.2:

69	Gudbrandsdalsost	36	26	15	ЛОЖЬ		
70	Outback Lager	15	15	30	ЛОЖЬ	15	225,00р.
71	Fluitemysost	21,5	26	0	ЛОЖЬ		
72	Mozzarella di Giovanni	34,8	14	0	ЛОЖЬ		
73	Rud Kaviar	15	101	5	ЛОЖЬ		
74	Longlife Tofu	10	4	5	ЛОЖЬ	1	10,00р.
75	Rheinbräu Klosterbier	7,75	125	25	ЛОЖЬ		
76	Lakkalikööri	18	57	20	ЛОЖЬ		
77	Original Frankfurter grüne Sojae	13	32	15	ЛОЖЬ		
Общая стоимость товаров на складе:			74,050,85р.				
Общая стоимость товаров к заказу:			3,633,45р.				

*Рис. 8.2 Последние строки с итоговыми значениями*

### Решение

- Создайте новый файл Excel, на вкладке **Разработчик** в группе **Элементы управления** нажмите кнопку **Вставить** и среди элементов ActiveX щелкните по элементу управления **Кнопка** и нарисуйте кнопку на листе Excel. Для определенности будем считать, что созданная кнопка занимает ячейки с A1 по E1 первого листа.
- Щелкните по кнопке **Свойства** (проверьте, что нажата кнопка **Режим конструктора**) и в появившемся окне **Properties** настройте для свойства **Caption** значение *Получить данные*. При необходимости воспользуйтесь свойством **Font**, чтобы настроить подходящий шрифт для вашей кнопки.



- Щелкните правой кнопкой мыши по созданной вами кнопке и в контекстном меню выберите **Исходный текст**. Откроется редактор Visual Basic с курсором ввода на месте события Click для вашей кнопки.
- В окне редактора кода в меню **Tools** выберите **References** и установите флажок напротив строки Microsoft ActiveX Data Objects 2.1 Library.

Код для события Click вашей кнопки **Получить данные** может быть таким:

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

' Создаем и настраиваем объект Connection
Dim cn As New ADODB.Connection
cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Nwind.mdb"
cn.Open

'Создаем и настраиваем объект Recordset
Dim rs As New ADODB.Recordset
rs.Open "SELECT[ProductID],[ProductName],[UnitPrice],[UnitsInStock],[ReorderLevel]," & _
"[Discontinued] FROM Products", cn
'На основе Recordset создаем объект QueryTable и вставляем его, начиная с 4-й строки
Dim QT1 As QueryTable
Set QT1 = QueryTables.Add(rs, Range("A4"))
QT1.Refresh

'Определяем количество записей в QueryTable
Dim nRowCount As Integer
Dim oRange As Range
Set oRange = QT1.ResultRange
nRowCount = oRange.Rows.Count

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

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

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

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

```

'Проходим циклом по всем ячейкам созданного диапазона
For Each oCell In oRange.Cells
'Получаем абсолютный номер строки в виде строковой переменной
sRowNumber = Replace(oCell.Address(True), "$G$", "")
'Проверяем определенные нами условия
If Range("E" & sRowNumber).Value > Range("D" & sRowNumber) And _
Range("F" & sRowNumber).Value = False Then
'Получаем значение для столбца G (заказ в штуках)
oCell.Value = (CInt(Range("E" & sRowNumber).Value) - CInt(Range("D" &
sRowNumber).Value))
'Получаем значение для столбца H (стоимость заказа)
cMoney = (CInt(Range("E" & sRowNumber).Value) - CInt(Range("D" & sRowNumber).Value))
* CCur(Range("C" & sRowNumber).Value)
'Записываем его в столбец H
Range("H" & sRowNumber).Value = cMoney
'Сразу плюсуем к итогу в рублях
cItogMoney = cItogMoney + cMoney
End If

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

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

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

```

## Литература

1. *Гарбер Г.З.* Основы программирование на Visual Basic и VBA в Excel 2007. – М.: СОЛОН-ПРЕСС, 2008. – 192 с.
2. *Назаров С.В. и др.* Программирование в пакетах MS Office: учеб. Пособие. – М.: Финансы и статистика, 2007.
3. *Слепцова Л.Д.* Программирование на VBA в Microsoft Office 2010. - – М. : Издательский дом “Вильямс”, 2010. – 432 с.
4. *Уокенбах, Джон* Профессиональное программирование на VBA в Excel 2010. : Пер. с англ. – М. : Издательский дом “Вильямс”, 2011. – 944 с.



В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

---

## КАФЕДРА ПРОГРАММНЫХ СИСТЕМ

Кафедра **Программных систем** входит в состав нового факультета **Инфокоммуникационные технологии**, созданного решением Ученого совета университета 17 декабря 2010 г. по предложению инициативной группы сотрудников, имеющих большой опыт в реализации инфокоммуникационных проектов федерального и регионального значения.

На кафедре ведется подготовка бакалавров и магистров по направлению **210700 «Инфокоммуникационные технологии и системы связи»:**

**210700.62.10 – ИНТЕЛЛЕКТУАЛЬНЫЕ  
ИНФОКОММУНИКАЦИОННЫЕ СИСТЕМЫ (Бакалавр)**

**210700.68.05 – ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ В  
ИНФОКОММУНИКАЦИЯХ (Магистр)**

Выпускники кафедры получают фундаментальную подготовку по: математике, физике, электронике, моделированию и проектированию инфокоммуникационных систем (ИКС), информатике и программированию, теории связи и теории информации.

В рамках профессионального цикла изучаются дисциплины: архитектура ИКС, технологии программирования, ИКС в Интернете, сетевые технологии, администрирование сетей Windows и UNIX, создание программного обеспечения ИКС, Web программирование, создание клиент-серверных приложений.

**Область профессиональной деятельности бакалавров и магистров включает:**

- сервисно-эксплуатационная в сфере современных ИКС;
- расчетно-проектная при создании и поддержке сетевых услуг и сервисов;
- экспериментально-исследовательская;

- организационно-управленческая – в сфере информационного менеджмента ИКС.

#### **Знания выпускников востребованы:**

- в технических и программных системах;
- в системах и устройствах звукового вещания, электроакустики, речевой, и мультимедийной информатики;
- в средствах и методах защиты информации;
- в методах проектирования и моделирования сложных систем;
- в вопросах передачи и распределения информации в телекоммуникационных системах и сетях;
- в методах управления телекоммуникационными сетями и системами;
- в вопросах создания программного обеспечения ИКС.

#### **Выпускники кафедры Программных систем обладают компетенциями:**

- проектировщика и разработчика структур ИКС;
- специалиста по моделированию процессов сложных систем;
- разработчика алгоритмов решения задач ИКС;
- специалиста по безопасности жизнедеятельности ИКС;
- разработчика сетевых услуг и сервисов в ИКС;
- администратора сетей: UNIX и Windows;
- разработчика клиентских и клиент-серверных приложений;
- разработчика Web – приложений;
- специалиста по информационному менеджменту;
- менеджера проектов планирования развития ИКС.

#### **Трудоустройство выпускников:**

- ОАО «Петербургская телефонная сеть»;
- АО «ЛЕНГИПРОТРАНС»;
- Акционерный коммерческий Сберегательный банк Российской Федерации;
- ОАО «РИВЦ-Пулково»;
- СПб ГУП «Петербургский метрополитен»;
- ООО «СоюзБалтКомплект»;
- ООО «ОТИС Лифт»;
- ОАО «Новые Информационные Технологии в Авиации»;
- ООО «Т-Системс СиАйЭс» и др.

**Кафедра** сегодня имеет в своем составе высококвалифицированный преподавательский состав, в том числе:

- 5 кандидатов технических наук, имеющих ученые звания профессора и доцента;
- 4 старших преподавателя;
- 6 штатных совмесителей, в том числе кандидатов наук, профессиональных IT-специалистов;

- 15 Сертифицированных тренеров, имеющих Западные Сертификаты фирм: Microsoft, Oracle, Cisco, Novell.

Современная техническая база; лицензионное программное обеспечение; специализированные лаборатории, оснащенные необходимым оборудованием и ПО; качественная методическая поддержка образовательных программ; широкие Партнерские связи существенно влияют на конкурентные преимущества подготовки специалистов.

Авторитет специализаций кафедры в области компьютерных технологий подтверждается Сертификатами на право проведения обучения по методикам ведущих Западных фирм - поставщиков аппаратного и программного обеспечения.

Заслуженной популярностью пользуются специализации кафедры ПС по подготовке и переподготовке профессиональных компьютерных специалистов с выдачей **Государственного Диплома** о профессиональной переподготовке по направлениям: **"Информационные технологии (инженер-программист)"** и **"Системный инженер"**, а также Диплома о дополнительном (к высшему) образовании с присвоением квалификации: **"Разработчик профессионально-ориентированных компьютерных технологий "**. В рамках этих специализаций высокопрофессиональные преподаватели готовят компетентных компьютерных специалистов по современным в России и за рубежом операционным системам, базам данных и языкам программирования ведущих фирм: Microsoft, Cisco, IBM, Intel, Oracle, Novell и др

Профессионализм, компетентность, опыт, и качество программ подготовки и переподготовки ИТ-специалистов на кафедре ПС неоднократно были удостоены **высокими наградами «Компьютерная Элита»** в номинации **лучший учебный центр России.**

#### **Партнеры:**

1. **Microsoft** Certified Learning Solutions;
2. **Novell** Authorized Education Center;
3. **Cisco** Networking Academy;
4. **Oracle** Academy;
5. **Sun Java** Academy и др;
6. **Prometric**;
7. **VUE**.

**Мы готовим квалифицированных инженеров в области инфокоммуникационных технологий с новыми знаниями, образом мышления и способностями быстрой адаптации к современным условиям труда.**

Ирина Станиславовна Осетрова  
Никита Алексеевич Осипов

## **Microsoft Visual Basic for Application**

**УЧЕБНОЕ ПОСОБИЕ**

В авторской редакции

Редакционно-издательский отдел НИУ ИТМО

Зав. РИО

Н.Ф. Гусарова

Лицензия ИД № 00408 от 05.11.99

Подписано к печати

Заказ №

Тираж 100 экз.

Отпечатано на ризографе

**Редакционно-издательский отдел**  
Санкт-Петербургского национального  
исследовательского университета  
информационных технологий, механики  
и оптики

197101, Санкт-Петербург, Кронверкский пр., 49

