Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Колледж электроники и бизнеса

Н.А. Уйманова, М.Г. Таспаева

ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Часть III

Рекомендовано к изданию Редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет» в качестве методических указаний для студентов, обучающихся по программам среднего профессионального образования по специальности 09.02.03 Программирование в компьютерных системах

Оренбург
2015

УДК 681.3.06:004.4(075.3)
ББК 32.973 Я73
У 35

Рецензент – кандидат педагогических наук, доцент А.Е.Шухман

Уйманова, Н.А.

У 35 Основы объектно-ориентированного программирования : методические указания к лабораторным работам: в 3 ч. Часть 3/ Н.А. Уйманова, М.Г. Таспаева; Оренбургский гос. ун-т. – Оренбург : ОГУ, 2015. – 74 с.

Методические указания предназначены для выполнения лабораторных работ, обеспечивающих учебный процесс по дисциплине «Основы объектно-ориентированного программирования», студентами 3 курса очной формы обучения специальности 09.02.03 «Программирование в компьютерных системах».

Методические указания составлены с учетом федерального государственного образовательного стандарта среднего профессионального образования по направлению подготовки дипломированных специалистов, утвержденного приказом № 804 от 28 июля 2014 года Министерством образования и науки Российской Федерации.

УДК 681.3.06:004.4(075.3)
ББК 32.973 Я73

Содержание

Введение

Учебная дисциплина «Основы объектно-ориентированного программирования» является дисциплиной из вариативной части ОПОП, обуславливающей знания для профессиональной деятельности выпускника.

Целью данного курса является формирование у будущего специалиста умений и навыков профессиональной направленности, написания программных продуктов, базирующихся на принципах объектно-ориентированного программирования, а так же сформировать представление о работе в среде визуального редактора.

У студентов необходимо сформировать такие умения и навыки, чтобы они могли в дальнейшем эффективно их использовать в своей профессиональной деятельности при программировании программных продуктов. Будущий специалист должен овладеть, прежде всего, базовыми принципами написания объектно-ориентированной программы, уметь работать в среде визуального программирования Delphi.

Задачами курса является:

- изучение основных понятий объектно-ориентированного программирования, их программное описание;
- изучение принципов объектно-ориентированного программирования;
- практическое освоение структуры написания классов с целью дальнейшего применения в профессиональной деятельности;
- изучение приемов организации распределения памяти для экземпляров класса;
- изучение технологии описания и применения виртуальных методов;
- выработка умений написания программных продуктов в среде визуального программирования Delphi.

Методические указания к лабораторным работам (часть 3) предназначены для проведения лабораторных занятий по дисциплине «Основы объектно-ориентированного программирования» по темам «Работа с диаграммами и таблицами в Delphi»,

«Исключительные ситуации», «Сервер автоматизации Microsoft Word» и «Работа с базами данных в Delphi».

Лабораторные работы предназначены для ознакомления с основными приемами работы с таблицами, выгрузкой данных в текстовый редактор. В заключительной серии работ рассматриваются вопросы практико-значимого характера, необходимые для дальнейшего проектирования самых простых программных средств.

1 Лабораторная работа №6. Построение диаграммы по данным таблицы. Обработка исключительных ситуаций в среде Delphi

1.1 Цель работы

Изучить основные приемы построения диаграмм по данным таблицы. Научиться обрабатывать исключительные ситуации.

1.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Создать новый проект delphi, осуществляющий расчет среднего арифметического введенных в таблицу элементов массива;

3 Спроектировать форму приложения, согласно рисунка 1, представленного в методических рекомендациях;

4 Обработать событие кнопки *расчет*.

5 Провести отладку и компиляцию проекта;

6 Сохранить проект на диске в директории lab6_1;

7 Создать новый проект delphi, осуществляющий построение круговой диаграммы по введенным данным о численности отличников каждой учебной группы 3 курса;

8 Спроектировать форму приложения, согласно рисунка 2, представленного в методических рекомендациях:

1) на форме разместить компонент *TStringGrid*, отображающий исходные данные;
2) разместить компонент *TChart*; вызвать редактор диаграммы, щелкнув два раза по компоненту; установить необходимые параметры диаграммы (см. методические рекомендации п. 1.5.2);

9 Обработать событие кнопки Button1, позволяющее отобразить на диаграмме введенные данные;

10 Провести отладку и компиляцию проекта;

11 Сохранить проект на диске в директории Lab6_2;

12 Создать новый проект Delphi, осуществляющий построение графика функции $y = \sin(x)$;

13 Спроектировать форму приложения, согласно рисунка 5, представленного в методических рекомендациях:

1) на форме разместить компонент *TStringGrid*, отображающий исходные данные;
2) разместить компонент *TChart*; вызвать редактор диаграммы, щелкнув два раза по компоненту; установить необходимые параметры диаграммы (см. методические рекомендации п. 1.5.2);

14 Обработать событие кнопки bitbtn1, позволяющее вычислить значение функции y;

15 Обработать событие кнопки bitbtn2, позволяющее построить график заданной функции;

16 Провести отладку и компиляцию проекта;

17 Сохранить проект на диске в директории Lab6_3;

18 Составить программу деления вещественных чисел. Программа должна выполнять обработку исключений с использованием конструкции try … except

19 Поместить на форму 3 компонента Edit, компонент Button;

20 У компонента Button организовать событие onclick согласно методическим рекомендациям п. 1.5.3.4;

21 Провести отладку и компиляцию проекта;

22 Сохранить проект на диске в директории Lab6_4.

Задание повышенного уровня:

23 создать проект в среде визуального программирования Delphi, отображающий график функции, согласно своего индивидуального задания (п.1.6);

24 Сохранить проект;

25 Провести отладку и компиляцию проекта;

26 Создать проект в среде визуального программирования Delphi, обрабатывающий исключительную ситуацию, согласно своего индивидуального задания (п.1.6);

27 Сохранить проект;

28 Провести отладку и компиляцию проекта;

29 Оформить отчет о проделанной работе;

30 Защитить работу.

1.3 Содержание отчета

1 Цель, ход работы;

2 Постановка задачи, листинг программного модуля, результат работы приложения;

3 Формулировка индивидуального задания;

4 Листинг программного модуля индивидуального задания, результат работы приложения;

5 Вывод.

1.4 Контрольные вопросы

1 Охарактеризуйте основные понятия объектно-ориентированного программирования;

2 Охарактеризуйте основные принципы объектно-ориентированного программирования;

3 Компонент *TStringGrid*, его свойства и события;

4 Компонент *TChart*, его свойства;

5 Виды ошибок, их характеристика;

6 Как осуществляется отладочное выполнение программы;

7 Виды исключений;

8 Обработка исключений. Глобальная и локальная обработка, их отличительные особенности;

9 Отличительные особенности конструкций *try…except* и *try…finally*.

1.5 Методические рекомендации

1.5.1 Таблицы в Delphi. Свойства и события класса *TStringGrid*

Компонент *StringGrid* находится на странице *Additional* палитры компонентов. *StringGrid* - компонент для отображения различных данных в табличной форме. Как следует из названия, ячейки компонента *StringGrid* Delphi могут содержать данные, имеющие тип *String*. Таблица *StringGrid* состоит из выделенных серым *FixedCols* и *FixedRows* - зафиксированных ячеек-заголовков, и обычных, белых ячеек. Содержимое *Fixed* ячеек недоступно редактированию, и меняется только программно.

Компонент *StringGrid* имеет возможность адресации каждой отдельной ячейки по номеру столбца и строки. Содержимое ячейки *(i, j)*, где где *i* - номер столбца, *j* - номер строки, имеет вид

StringGrid1.Cells[i, j]

и доступно как для чтения, так и для записи. Здесь номера столбцов (*i*) и строк (*j*) отсчитываются от *0***.**

В таблице 1 перечислены некоторые свойства компонента *StringGrid*.

Таблица 1 – Свойства компонента *StringGrid*

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к свойствам компонента
ColCount	Количество колонок таблицы
RowCount	Количество строк таблицы
Cells	Соответствующий таблице двумерный массив. Ячейка таблицы, находящаяся на пересечении столбца номер *col* и строки номер *row* определяется элементом *cells* [*col*, *row*]
FixedCols	Количество зафиксированных колонок слева таблицы
FixedRows	Количество зафиксированных строк сверху таблицы
Font	Шрифт отображения содержимого ячеек
Options.goEditing	Возможность редактировать содержимое ячейки с клавиатуры
Options.goRowSizing	Возможность менять высоту строк мышкой
Options.goColSizing	Возможность менять ширину столбцов мышкой
Options.goRowMoving	Возможность менять номер строки, то есть перемещать её, мышкой
Options.goColMoving	Возможность менять номер столбца, то есть перемещать его, мышкой

В таблице 2 перечислены некоторые события компонента *StringGrid*.

Таблица 2 – События компонента *StringGrid*

Событие	Описание
OnClick	Происходит в момент щелчка по компоненту StringGrid
OnEnter	Происходит в момент получения компонентом StringGrid фокуса ввода. Происходить это может как вручную, кликом мышки или нажатием клавиши Tab, так и программно, с помощью оператора
OnSelectCell	Происходит в момент перехода фокуса ввода в одну из ячеек таблицы StringGrid, однако ещё до непосредственного перехода
OnKeyPress	Происходит при нажатии клавиши на клавиатуре

В качестве примера использования компонента *StringGrid* для ввода массива рассмотрим программу, которая вычисляет среднее арифметическое значение элементов массива. Внешний интерфейс программы приведен на рисунке 1. Компонент *StringGrid* используется для ввода массива, компоненты *Label1* и *Label2* — для вывода пояснительного текста и результата расчета соответственно, *Button1*— для запуска процесса расчета.

В инспекторе объектов у компонента *StringGrid1* изменяем следующие свойства: *FixedRows = 1; FixedCols = 0; RowCount = 2; Options.goEditing = true.*

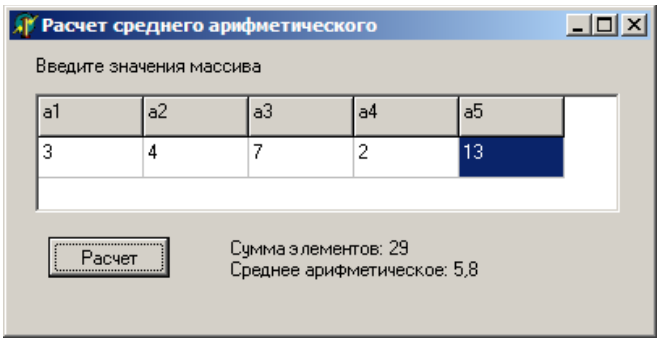


Рисунок 1 – Внешний интерфейс приложения

В разделе переменных *Var* опишем переменную *i* – счетчик элементов массива

var

```
  Form1: TForm1;
  i:integer;
```

Зададим значения элементов строки-заголовка таблицы (обработчик события *OnCreate* для компонента *Form1*)

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  for i:=1 to 5 do
    StringGrid1.Cells[i-1,0]:='a'+IntToStr(i);
end;
```

Оформим обработчик события *ButtonlClick* кнопки *Button1*

```
procedure TForml.ButtonlClick(Sender: TObject);
var
a : array[1..5] of integer; // массив
summ: integer; // сумма элементов
sr: real; // среднее арифметическое
i: integer; // индекс
begin
// ввод массива
// считаем, что если ячейка пустая, то соответствующий
// ей элемент массива равен нулю
for i:= 1 to 5 do
if Length(StringGridl.Cells[i-1, 0]) <>0
then a[i] := StrToInt(StringGridl.Cells[i-1,0])
else a[i] := 0;
// обработка массива
summ := 0;
for i :=1 to 5 do
summ := summ + a[i]; sr := summ / 5;
// вывод результата Label2.Caption :=
'Сумма элементов: ' + IntToStr(summ)
```

```
+ #13+ 'Среднее арифметическое: ' + FloatToStr(sr);
end;
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  a:array[1..5] of integer;
  summ:integer;
  sr:real;
begin
  // ввод массива
  // считаем, что если ячейка пустая, то соответствующий
  // ей элемент массива равен нулю
   for i:= 1 to 5 do
   if Length(StringGrid1.Cells[i-1, 0]) <>0
   then a[i] := StrToInt(StringGrid1.Cells[i-1,1])
   else a[i] := 0;
  // обработка массива
   summ := 0;
   for i :=1 to 5 do
    summ := summ + a[i];
    sr := summ / 5;
 // вывод результата
   Label2.Caption := 'Сумма элементов: ' + IntToStr(summ)
   + #13+ 'Среднее арифметическое: ' + FloatToStr(sr);
end;
```

1.5.2 Построение диаграммы по данным таблицы

Компонент *Chart*, расположенный во вкладке *Additional* палитры компонентов *Delphi*, позволяет строить различные диаграммы и графики.

Является контейнером объектов *Series* типа *TChartSeries* – серий данных, характеризующихся различными стилями отображения. Каждая серия будет соответствовать одной кривой на графике.

Основные свойства компонента *TChart* представлены в таблице 3.

Таблица 3 – Свойства компонента *TChart*

Свойство	Описание
1	2
AllowPanning	Определяет возможность пользователя прокручивать наблюдаемую часть графика во время выполнения, нажимая правую кнопку мыши: pmNone – прокрутка запрещена; pmHorizontal – разрешена прокрутка только в горизонтальном направлении; pmVertical – только в вертикальном pmBoth – в обоих напрвлениях
AllowZoom	Позволяет пользователю изменять во время выполнения масштаб изображения, вырезая фрагменты диаграммы или графика курсором мыши.
Title	Определяет заголовок диаграммы
Foot	Определяет подпись под диаграммой. По умолчанию отсутствует. Текст подписи определяется подсвойством Text
Frame	Определяет рамку вокруг диаграммы
Legend	Легенда диаграммы – список обозначений

Продолжение таблицы 3

1	2

MarginLeft, *MarginRight* *MarginTop* *MarginButtom*	Значения левого, правого, верхнего и нижнего полей
ButtomAxis *LeftAxis* *RightAxis*	Эти свойства определяют характеристики соответственно нижней, левой и правой осей. Задание этих свойств имеет смысл для графиков и некоторых других типов диаграмм
LeftWall *ButtomWall* *BackWall*	Эти свойства определяют характеристики соответственно левой, нижней и задней граней области трехмерного отображения графика
SeriesList	Список серий данных, отображаемых в компоненте
View3d	Разрешает или запрещает трехмерное отображение диаграммы
View3dOptions	Характеристики трехмерного отображения
Chart3DPersent	Масштаб трехмерности (толщина диаграммы, ширина лент графика)

В качестве примера разработаем приложение, основной функцией которой является построение диаграммы по введенным в таблицу данным о численности отличников 3 курса специальности «Программирование в компьютерных системах».

Примерный внешний интерфейс программы представлен на рисунке 2. Компонент *StringGrid* используется для ввода массива, *Button1*— для формирования диаграммы по исходным данным.

В инспекторе объектов у компонента StringGrid1 изменяем следующие свойства: *FixedRows* = 1; *FixedCols* = 1; *RowCount* = 5; *ColCount* = 2; *Options.goEditing = true*.

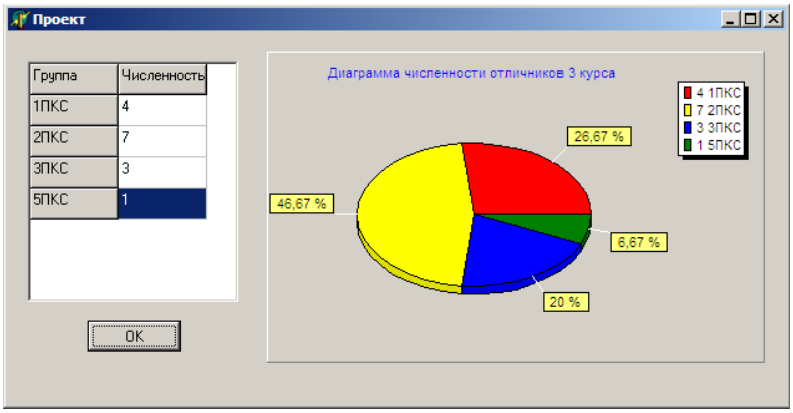


Рисунок 2 – Внешний интерфейс приложения

Выведем круговую диаграмму. Перейти в редактор диаграмм Chart1 можно следующими способами:

– кнопкой с многоточием рядом с названием свойства в инспекторе объектов;
– двойным щелчком на компоненте *Chart* при проектировании формы;
– выбором команды *Edit Chart* в контекстном меню компонента *Chart* при проектировании формы.

На закладке *Chart*, на вкладке *Series* щелкнуть на кнопке *Add* – добавить серию. Вы попадаете в окно, в котором можно выбрать тип диаграммы или графика. В данном случае выберем *Pie* – круговая диаграмма.

Закладка *Titles* позволяет задавать заголовок диаграммы (Диаграмма численности отличников)

Закладка *Legend* позволяет задавать позволяет задавать параметры отображения легенды легенды диаграммы (списка обозначений) или вообще убирать ее с экрана.

Закладка *Panel* определяет вид панели, на которой отображается диаграмма.

Закладка *3D* позволяет определить внешний вид диаграммы: сдвиг, наклон, толщину и т.д.

На закладке *Series*, на вкладке *Marks* в качестве значения *Style* выберем *Percent* (для отображения процентного соотношения отличников для каждой группы относительно общей численности).

Опишем обработчик события *OnCreate* компонента *Form1*. Задаем значения фиксированным столбцу и строке таблицы.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[0,0]:='Группа';
  StringGrid1.Cells[1,0]:='Численность';
  StringGrid1.Cells[0,1]:='1ПКС';
  StringGrid1.Cells[0,2]:='2ПКС';
  StringGrid1.Cells[0,3]:='3ПКС';
  StringGrid1.Cells[0,4]:='5ПКС';
end;
```

Для задания отображаемых значений надо использовать методы серий *Series*:

1 *Clear* – очищает серию от занесенных ранее данных;

2 *Add* – позволяет добавить в диаграмму новую точку:

```
Add(Const AValue:Double; Const ALabel:String; AColor:TColor)
```

Параметр *AValue* соответствует добавляемому значению, параметр *ALabel* – название, которое буде отображаться на диаграмме и в легенде, параметр *AColor* – цвет. Параметр *ALabel* необязательный, его можно задавать пустым;

3 *AddXY* – позволяет добавить новую точку в график функции:

```
AddXY(Const AXValue, AYValue: Double; Const ALabel: String; AColor: TColor);
```

Параметры *AXValue* и *AYValue* соответствуют аргументу и функции, параметры *ALabel* и *AColor* – те же, что и в методе *Add*.

Обработаем событие *OnClick* объекта *Button1*, позволяющее отобразить данные из таблицы на диаграмме.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with series1 do
  begin
    Add(StrToInt(StringGrid1.Cells[1,1]),StringGrid1.Cells[0,1],clRed);
    Add(StrToInt(StringGrid1.Cells[1,2]),StringGrid1.Cells[0,2],clYellow);
```

```
    Add(StrToInt(StringGrid1.Cells[1,3]),StringGrid1.Cells[0,3],clBlue);
    Add(StrToInt(StringGrid1.Cells[1,4]),StringGrid1.Cells[0,4],clGreen);
  end;
end;
```

В качестве второго задания построим график функции $y = \sin(x)$. Внешний интерфейс программы представлен на рисунке 3.

Компонент *StringGrid* используется для ввода массива, *BitBtn1*— для расчета значений функции по известному аргументу *x*, *BitBtn2*— для формирования графика заданной функции, *Chart1* – для отображения графика функции.

В инспекторе объектов у компонента *StringGrid1* изменяем следующие свойства: *FixedRows* = 0; *FixedCols* = 1; *RowCount* = 2; *ColCount* = 6; *Options.goEditing = true*.

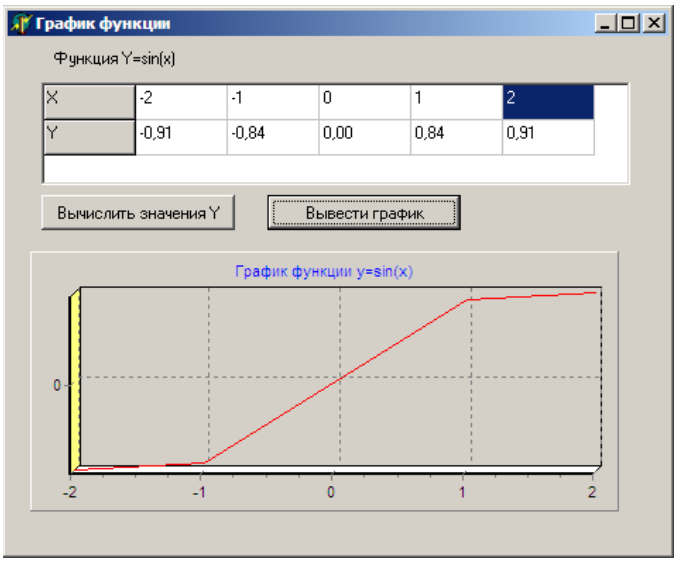


Рисунок 3 – Внешний интерфейс программы

В разделе реализации описываем функцию зависимости *y* от *x*.

```
Function f(xx:real) :real;
begin
  f:=Sin(xx);
end;
```

В обработчике события *OnCreate* компонента *Form1* задаем значения ячеек фиксированного столбца таблицы.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[0,0]:='X';
  StringGrid1.Cells[0,1]:='Y';
end;
```

Формируем обработчик события *OnClick* кнопки *BitBtn1*, в котором осуществляется расчет значений функции по введенным значениям *x*. Процедура *FloatToStrF* позволяет ограничить количество знаков после запятой вещественного числа.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 for i:=1 to 5 do
 begin
  x[i]:=StrToFloat(StringGrid1.Cells[i,0]);
  y[i]:=f(x[i]);
  StringGrid1.Cells[i,1]:=FloatToStrF(y[i],ffFixed,4,2);
 end;
end;
```

Формируем обработчик события *OnClick* кнопки *BitBtn2*, в котором осуществляется построение графика функции путем последовательного добавления точек.

```
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
   for i:=1 to 5 do
   series1.AddXY(x[i],y[i],FloatToStr(x[i]),clRed);
end;
```

1.5.3 Обработка исключительных ситуаций в среде Delphi

На этапе выполнения Delphi порождает исключения, когда какой-либо процесс идет неправильно. Если код подпрограммы написан соответствующим образом, он может распознать возникшую проблему и попытаться ее решить; в противном случае исключение передается в код, который вызвал вашу подпрограмму и т.д. В конечном счете, если никто не обработал исключение, его обрабатывает Delphi, выводя на экран стандартное сообщение об ошибке и пытаясь продолжить выполнение программы.

1.5.3.1 Виды ошибок

В процессе разработки и выполнения программы возникают ошибки:

синтаксические;

логические;

динамические.

Синтаксические ошибки вызываются нарушением синтаксиса языка, они выявляются и устраняются при компиляции программы. Их обнаруживает компилятор, выдавая сообщения и указывая в тексте программы место, где возникла ошибка. Например, в условной инструкции

If length(edit1.text) = 0 then edit1.text = ‘Нет имени’;

допущена ошибка — в записи операции присваивания отсутствует знак двоеточия (:). При ее обнаружении в ходе компиляции будет выдано соответствующее сообщение.

Логические ошибки являются следствием реализации неправильного алгоритма и проявляются при выполнении программы. Их наличие обычно не приводит к выдаче сообщений или прекращению работы всего приложения, однако программа будет работать некорректно и выдавать неправильные, результаты.

Фрагмент программы, в котором вычисляется сумма элементов массива *MAS*:

Sum:= 1;

For i:=1 to 50 do sum:=sum+MAS[i];

Перед началом суммирования значение суммы должно обнуляться, однако в программе допущена ошибка: вместо нуля переменной sum присваивается начальное значение, равное единице. Такая ошибка не приведет к прекращению выполнения программы, однако получаемый при суммировании результат будет неверен.

Динамические ошибки возникают при выполнении программы и являются следствием неправильной работы инструкций, процедур, функций или методов программы, а также операционной системы. Динамические ошибки называют также ошибками времени выполнения (*Runtime errors*). Например, в инструкции присваивания

I:= count/number

во время выполнения программы возможно появление ошибки, если переменная number будет иметь нулевое значение.

При отладке программ с целью выявления динамических ошибок удобно задавать отладочное (пошаговое или трассировочное) выполнение программы с использованием окон просмотра (*Watch list*). В окне просмотра можно указать выражения, имена переменных или свойств объекта, изменение значений которых требуется проконтролировать.

Для отладочного выполнения программы с помощью меню Run (Выполнение) можно использовать следующие варианты:

команда *Step Over* (Шаг с обходом) предписывает выполнение од-ной строки кода программы с обходом процедур (процедура выполняется как единый модуль);

команда *Trace Into* (Трассирование до) предписывает выполнение одной строки кода программы с заходом в процедуры и их последующим построчным выполнением;

команда *Trace to next source line* (Трассирование до следующей строки кода) предписывает выполнение программы с остановкой на следующей выполнимой строке кода программы;

команда *Run to cursor* (Выполнение до курсора) задает выполнение загруженной программы до места размещения курсора в Редакторе кода;

команда *Run until Return* (Выполнение до возврата) задает выполнение программы до момента возврата из текущей процедуры.

Для добавления очередного контролируемого выражения в окно про-смотра служит команда *Run/Add Watch*. После ее выполнения открывается диалоговое, *Watch Properties* (Свойства просмотра) для задания свойств контролируемого выражения. В поле *Expression* (Выражение) задается выражение для контроля его значения, в поле *Group name* (Имя группы) можно задать имя группы, в которую будет помещено контролируемое выражение. С помощью группы переключателей в нижней части диалогового окна выбирается формат отображения значения контролируемого выражения.

Для обработки динамических ошибок введено понятие исключения, ко-торое представляет собой нарушение условий выполнения программы, вызывающее прерывание или полное прекращение ее работы. Обработка исключения состоит в нейтрализации вызвавшей его динамической ошибки.

Исключения могут возникать по различным причинам, например, в случае нехватки памяти, из-за ошибки преобразования, в результате выполнения вычислений и т. д. В любом случае независимо от источника ошибки приложение информируется (получает сообщение) о его возникновении. Исключение остается актуальным до тех пор, пока не будет обработано глобальным обработчиком или локальными процедурами.

1.5.3.2 Глобальная обработка

Для обработки исключений в приложении есть один глобальный обработчик и несколько специализированных процедур-обработчиков, реагирующих на соответствующие исключения. Каждое исключение обрабатывает свой специализированный локальный обработчик. Исключение, не имеющее своего

локального обработчика, обрабатывается глобальным обработчиком приложения.

Механизм глобальной обработки исключений реализуется через объект *Application*, который есть в любом приложении.

Программист может выполнить полную обработку исключений, создав собственный глобальный обработчик события *OnException*. Для этого удобно использовать компонент *ApplicationEvents*.

Событие *OnException* имеет тип *TExceptionEvent*, который описан следующим образом:

Type TExceptionEvent = procedure (Sender: TObject; E: Exception) of Object

Глобальный обработчик может содержать код, зависящий от особенностей конкретной программы, например, освобождающий память или закрывающий рабочие файлы.

1.5.3.3 Локальная обработка

Чтобы сделать возможным использование локальных (специализированных) обработчиков исключений, в состав языка введены две конструкции*: try ...finally и try ... except*. Обе конструкции имеют похожий синтаксис, но разное назначение. Блоки у включают в себя инструкции программы

Выбор конструкции зависит от применяемых инструкций программы и действий, выполняемых при возникновении ошибки. Конструкции try могут содержать одну или более инструкций, а также быть вложенными друг в друга.

Конструкция *try ...finally* состоит из двух блоков (try и finally) и имеет следующую форму:

```
try
// Инструкции, выполнение которых может вызвать ошибку
finally
// Инструкции, которые должны быть выполнены даже в случае ошибки
end;
```

Конструкция *try ...finally* работает так: если в любой из инструкций блока try возникает исключение, то управление передается первой инструкции блока finally. Если же исключение не возникло, то последовательно выполняются все инструкции обоих блоков.

Так как конструкция *try ...finally* не ликвидирует исключительную ситуацию, в приведенной процедуре при возникновении исключения глобальный обработчик выдаст сообщение о характере ошибки.

Конструкция *try ... except* также состоит из двух блоков (*try и except*) и имеет следующую форму:

Try

{Инструкции, выполнение которых может вызвать ошибку}

Except

{Инструкции, которые должны быть выполнены в случае ошибки}

End;

В отличие от предыдущей, данная конструкция применяется для перехвата исключения и предоставляет возможность его обработки.

Конструкция *try ... except* работает так: если в инструкциях блока *try* возникает исключение, то управление передается первой инструкции блок *except*. Если же исключение не возникло, то инструкции блока *except* не выполняется. При появлении исключения инструкции блока *except* могут ликвидировать исключительную ситуацию и восстановить работоспособность программы. Для исключений, обрабатываемых в конструкции *try ... except*, глобальный обработчик не вызывается, а обработку ошибок должен обеспечить программист.

Блок *except* можно разбить на несколько частей с помощью конструкций *on… do,* позволяющих анализировать класс исключения для его более удобной и полной обработки. Конструкция *on…do* применяется в случаях, когда действия по обработке исключения зависят от класса исключения, и имеет следующую форму:

On {идентификатор: класс исключения} do

{инструкции обработки исключения этого класса};

Else {инструкции}

В инструкции *on* класс возникшего исключения сравнивается с указанным классом исключения. В случае совпадения классов выполняются инструкции после слова *do*, реализующие обработку этого исключения.

Идентификатор (произвольное имя, заданное программистом) является необязательным элементом и может отсутствовать, при этом не ставится и разделительный знак двоеточия (:). Идентификатор — это локальная переменная, представляющая собой экземпляр класса исключения, который можно использовать для доступа к объекту возникшего исключения. Эта переменная доступна только внутри “своей” конструкции *on…do*.

Если класс возникшего исключения не совпадает с проверяемым классом, то выполняются инструкции после слова *else*. Блок *else* является необязательным и может отсутствовать.

Если в блоке *except* расположено несколько конструкций *on…do*, то *else* располагается в конце блока и относится ко всей совокупности конструкций *on… do.*Следующая после слова *else* инструкция выполняется в том случае, если обработка исключения не была осуществлена ни одной из инструкций, расположенных в любой из конструкций *do* блока. Инструкции, следующие после слов *do и else*, могут быть составными.

Конструкции *try* могут быть вложенными и размещаться одна в другой. При этом внешняя и внутренняя конструкции могут иметь любой из двух рассмотренных видов. Обязательным условием является то, что внутренний блок должен полностью размещаться во внешнем блоке. Например:

try
{Инструкции}
try
{Инструкции}
finally
{Инструкции}
end;
except

{Инструкции}

end;

или

try

{Инструкции}

try

{Инструкции}

except

{Инструкции}

end;

finally

{Инструкции}

end;

Если какие-либо действия должны быть выполнены независимо от того, произошла ошибка или нет, то удобно использовать конструкцию try…finally. Однако, как отмечалось, эта конструкция не обрабатывает исключение, а лишь в некоторой степени смягчает его последствия. Если же требуется произвести и локальную обработку исключения, то можно включить конструкцию try…finally в конструкцию try…except. При возникновении исключения это позволяет выполнить обязательные инструкции блока finally и обработать исключение инструкциями блока except.

1.5.3.4 Классы исключений

Исключения в Delphi являются объектами. Базовым классом для всех исключений служит класс Exception, описываемый в модуле SysUtils. Потомки этого класса содержат большое количество исключений, которые могут возникнуть в процессе выполнения приложения.

Виды исключений:

- EAbort – «скрытое» исключение. Используйте его тогда, когда хоти-те

прервать тот или иной процесс с условием, что пользователь программы не должен видеть сообщения об ошибке. Для повышения удобства использования в модуле SysUtils предусмотрена процедура Abort, опре-деленная, как:

```
procedure Abort;
begin
raise EAbort.CreateRes(SOperationAborted) at ReturnAddr;
end;
```

- EComponentError - вызывается в двух ситуациях: при попытке регистрации компоненты за пределами процедуры Register; когда имя компоненты не уникально или не допустимо;

- EConvertError - происходит в случае возникновения ошибки при выполнении функций StrToInt и StrToFloat, когда конвертация строки в соответствующий числовой тип невозможна;

- EInOutError - происходит при ошибках ввода/вывода при включенной директиве {$I+};

- EIntError - предок исключений, случающихся при выполнении целочисленных операций;

- EDivByZero - вызывается в случае деления на ноль, как результат RunTime Error 200;

- EIntOverflow - вызывается при попытке выполнения операций, приводящих к переполнению целых переменных, как результат RunTime Error 215 при включенной директиве {$Q+};

- ERangeError - вызывается при попытке обращения к элементам массива по индексу, выходящему за пределы массива, как результат RunTime Error 201 при включенной директиве {$R+};

- EInvalidCast - происходит при попытке приведения переменных одного класса к другому классу, несовместимому с первым (например, приведение переменной типа TListBox к TMemo);

- EInvalidGraphic - вызывается при попытке передачи в LoadFromFile файла, несовместимого графического формата;

- EInvalidGraphicOperation - вызывается при попытке выполнения операций, неприменимых для данного графического формата (например, Resize для TIcon);

- EInvalidObject - реально нигде не используется, объявлен в Controls.pas;

- EInvalidOperation - вызывается при попытке отображения или обращения по Windows-обработчику (handle) контрольного элемента, не имеющего владельца (например, сразу после вызова MyControl:=TListBox.Create(...) происходит обращение к методу Refresh);

- EInvalidPointer - происходит при попытке освобождения уже освобожденного или еще неинициализированного указателя, при вызове Dispose(), FreeMem() или деструктора класса;

- EListError - вызывается при обращении к элементу наследника TList по индексу, выходящему за пределы допустимых значений (например, объект TStringList содержит только 10 строк, а происходит обращение к одиннадцатому);

- EMathError - предок исключений, случающихся при выполнении операций с плавающей точкой;

- EInvalidOp - происходит, когда математическому сопроцессору передается ошибочная инструкция. Такое исключение не будет до конца обработано, пока Вы контролируете сопроцессор напрямую из ассемблерного кода;

- EOverflow - происходит как результат переполнения операций с плавающей точкой при слишком больших величинах. Соответствует RunTime Error 205;

- Underflow - происходит как результат переполнения операций с плавающей точкой при слишком малых величинах. Соответствует RunTime Error 206;

- EZeroDivide - вызывается в результате деления на ноль;

- EMenuError - вызывается в случае любых ошибок при работе с пунктами меню для компонент TMenu, TMenuItem, TPopupMenu и их наследников;

- EOutlineError - вызывается в случае любых ошибок при работе с TOutLine и любыми его наследниками;

- EOutOfMemory - происходит в случае вызовов New(), GetMem() или конструкторов классов при невозможности распределения памяти. Соответствует RunTime Error 203;

- EOutOfResources - происходит в том случае, когда невозможно выполнение запроса на выделение или заполнение тех или иных Windows ресурсов (например таких, как обработчики - handles);

- EParserError - вызывается когда Delphi не может произвести разбор и перевод текста описания формы в двоичный вид (часто происходит в случае исправления текста описания формы вручную в IDE Delphi);

- EPrinter - вызывается в случае любых ошибок при работе с принтером;

- EProcessorException - предок исключений, вызываемых в случае прерывания процессора- hardware breakpoint. Никогда не включается в DLL, может обрабатываться только в «цельном» приложении;

- EBreakpoint - вызывается в случае останова на точке прерывания при отладке в IDE Delphi. Среда Delphi обрабатывает это исключение самостоятельно;

- EFault - предок исключений, вызываемых в случае невозможности обработки процессором тех или иных операций;

- EGPFault - вызывается, когда происходит «общее нарушение защиты» - General Protection Fault. Соответствует RunTime Error 216;

- EInvalidOpCode - вызывается, когда процессор пытается выполнить недопустимые инструкции;

- EPageFault - обычно происходит как результат ошибки менеджера памяти Windows, вследствие некоторых ошибок в коде Вашего приложения. После такого исключения рекомендуется перезапустить Windows;

- EStackFault - происходит при ошибках работы со стеком, часто вследствие некорректных попыток доступа к стеку из фрагментов кода на ассемблере. Компиляция Ваших программ со включенной проверкой работы со стеком {$S+} помогает отследить такого рода ошибки;

- ESingleStep - аналогично EBreakpoint, это исключение происходит при пошаговом выполнении приложения в IDE Delphi, которая сама его и обрабатывает;

- EPropertyError - вызывается в случае ошибок в редакторах свойств, встраиваемых в IDE Delphi. Имеет большое значение для написания надежных property editors. Определен в модуле DsgnIntf.pas;

- EResNotFound - происходит в случае тех или иных проблем, имею-щих место при попытке загрузки ресурсов форм - файлов .DFM в режиме дизайнера. Часто причиной таких исключений бывает нарушение соответствия между определением класса формы и ее описанием на уровне ресурса (например,вследствие изменения порядка следования полей-ссылок на компоненты, вставленные в форму в режиме дизайнера);

- EStreamError - предок исключений, вызываемых при работе с потоками;

- EFCreateError - происходит в случае ошибок создания потока (например, при некорректном задании файла потока);

- EFilerError - вызывается при попытке вторичной регистрации уже зарегистрированного класса (компоненты). Является, также, предком специализированных обработчиков исключений, возникающих при работе с классами компонент;

- EClassNotFound - обычно происходит, когда в описании класса формы удалено поле-ссылка на компоненту, вставленную в форму в режиме дизайнера. Вызывается, в отличие от EResNotFound, в RunTime;

- EInvalidImage - вызывается при попытке чтения файла, не являющегося ресурсом, или разрушенного файла ресурса, специализированными функциями чтения ресурсов (например, функцией ReadComponent);

- EMethodNotFound - аналогично EClassNotFound, только при несоответствии методов, связанных с теми или иными обработчиками событий;

- EReadError - происходит в том случае, когда невозможно прочитать значение свойства или другого набора байт из потока (в том числе ресурса);

- EFOpenError - вызывается когда тот или иной специфированный поток не может быть открыт (например, когда поток не существует);

- EStringListError - происходит при ошибках работы с объектом TStringList (кроме ошибок, обрабатываемых TListError).

Составить программу деления вещественных чисел. Программа должна выполнять обработку исключений с использованием конструкции try … except.

Организовать событие OnClick. Внутри процедуры прописать следующий код.

```
try
 x:= StrToInt(Edit1.Text);
 y:= StrToInt(Edit2.Text);
 res:= x/y;
 Edit3.Text:=FloatToStr(res);
except
 on EZeroDivide do
  begin
    MessageDlg('Попытка деления на ноль!', mtError, [mbOK], 0);
    edit2.SetFocus;
    edit3.Text:='Ошибка!';
  end;
on EconvertError do
begin
        MessageDlg('Ошибка  преобразования!'+#10#13+EO.message,  mtError,
[mbOK], 0);
   edit2.SetFocus;
   edit3.Text:='Ошибка!';
end;
else begin
  MessageDlg('Ошибка не идентифицирована!', mtWarning, [mbOK], 0);
  edit2.SetFocus;
  edit3.Text:='Ошибка!';
end;
end;
end;
```

1.6 Индивидуальные задания

1 Выполнить задания ниже:

1) Построить график функции $y = 3\cos 2x$, используя компоненты *TStringGrid* и *TChart*;

2) Составить программу деления вещественных чисел. Программа должна выполнять обработку исключений с использованием конструкции *try ... finally* или *try ... except*, и выдавать следующее сообщение о характере ошибки: *«не введено число»*

2 Выполнить задания ниже:

1) Построить график функции $y = 2x + x^2$, используя компоненты TStringGrid и TChart;

2) Составить программу, в которой при нажатии кнопки в форме отображается изображение из файла. Программа должна выполнять обработку исключений с использованием конструкции try … finally или try … except, и выдавать следующее сообщение о характере ошибки: «ошиб*ка открытия файла*»

3 Выполнить задания ниже:

1) Построить график функции $y = e^x + 3$, используя компоненты TStringGrid и TChart;

2) Составить программу деления вещественных чисел. Программа должна выполнять обработку исключений с использованием конструкции try … finally или try … except, и выдавать следующее сообщение о характере ошибки: «*введено слишком длинное число*»

4 Выполнить задания ниже:

1) Построить график функции $y = x^2 - 2$, используя компоненты TStringGrid и TChart;

2) Составить программу, в которой при нажатии кнопки в форме отображается изображение из файла. Программа должна выполнять обработку исключений с использованием конструкции *try ... finally* или *try ... except*, и выдавать следующее сообщение о характере ошибки: «*ошибка чтения файла*»

5 Выполнить задания ниже:

1) Построить график функции $y = 2\cos x - 1$, используя компоненты TStringGrid и TChart;
2) Составить программу деления вещественных чисел. Программа должна выполнять обработку исключений с использованием конструкции *try ... finally* или *try ... except*, и выдавать следующее сообщение о характере ошибки: «*деление на ноль*»

6 Выполнить задания ниже:
1) Построить график функции $y = 2x + 3$, используя компоненты TStringGrid и TChart;
2) Составить программу, в которой при нажатии кнопки в форме отображается изображение из файла. Программа должна выполнять обработку исключений с использованием конструкции try … finally или try … except, и выдавать следующее сообщение о характере ошибки: «ошибка графики»

7 Выполнить задания ниже:
1) Построить график функции $y = e^{4x}$, используя компоненты TStringGrid и TChart;
2) Составить программу деления вещественных чисел. Программа должна выполнять обработку исключений с использованием конструкции try … finally или *try ... except*, и выдавать следующее сообщение о характере ошибки: «*ошибка преобразования*»

8 Выполнить задания ниже:
1) Построить график функции $y = 4 - x^2$, используя компоненты TStringGrid и TChart;
2) Составить программу, в которой при нажатии кнопки в форме отображается изображение из файла. Программа должна выполнять обработку исключений с использованием конструкции *try ... finally* или *try ... except*, и выдавать следующее сообщение о характере ошибки: *«неправильный графический формат»*

9 Выполнить задания ниже:
1) Построить график функции $y = \cos x + 2$, используя компоненты TStringGrid и TChart;
2) Составить программу деления вещественных чисел. Программа должна выполнять обработку исключений с использованием конструкции

try … finally или *try ... except*, и выдавать следующее сообщение о характере ошибки: «*ошибка не идентифицирована*»

10 Выполнить задания ниже:

1) Построить график функции $y = x - 8$, используя компоненты TStringGrid и TChart;

2) Составить программу, в которой при нажатии кнопки в форме отображается изображение из файла. Программа должна выполнять обработку исключений с использованием конструкции try … finally или try … except, и выдавать следующее сообщение о характере ошибки: «ошибка не идентифицирована»

11 Выполнить задания ниже:

1) Построить график функции $y = 4x + x^2$, используя компоненты TStringGrid и TChart;

2) Составить программу, вычисляющую тангенс угла. Программа должна выполнять обработку исключений с использованием конструкции try … finally или try … except, и выдавать следующее сообщение о характере ошибки: «*тангенс угла 90 градусов не существует*»

12 Выполнить задания ниже:

1) Построить график функции $y = e^{2x} + 1$, используя компоненты TStringGrid и TChart;

2) Составить программу, вычисляющую тангенс угла. Программа должна выполнять обработку исключений с использованием конструкции try … finally или try … except, и выдавать следующее сообщение о характере ошибки: «*значение угла не введено*»

13 Выполнить задания ниже:

1) Построить график функции $y = 3x^2$, используя компоненты TStringGrid и TChart;

2) Составить программу, вычисляющую тангенс угла. Программа должна выполнять обработку исключений с использованием конструкции try … finally или try … except, и выдавать следующее сообщение о характере ошибки: «*введено не число*»

14 Выполнить задания ниже:

1) Построить график функции $y = 4x + 3$, используя компоненты TStringGrid и TChart;

2) Составить программу, вычисляющую тангенс угла. Программа должна выполнять обработку исключений с использованием конструкции try … finally или try … except, и выдавать следующее сообщение о характере ошибки: «*введено слишком большое значение угла*»

15 Выполнить задания ниже:

1) Построить график функции $y = x^2 + 2$, используя компоненты TStringGrid и TChart;

2) Составить программу, вычисляющую тангенс угла. Программа должна выполнять обработку исключений с использованием конструкции *try ... finally* или *try ... except*, и выдавать следующее сообщение о характере ошибки: «ошибка не идентифицирована»

2 Лабораторная работа №7. Сервер автоматизации MS Word

2.1 Цель работы

Изучить процедуры работы с сервером автоматизации. Научиться осуществлять взаимодействие приложения Delphi с MS Word.

2.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Создать новый проект delphi, осуществляющий занесение данных с формы delphi в приложение ms word;

3 Спроектировать форму приложения, согласно рисунка 5, представленного в методических рекомендациях;

4 Для компонента pagecontrol создать две вкладки (п.к.м. по компоненту page-control->new page, изменить свойство caption каждой вкладки);

5 Разместить иконки на вкладках компонента pagecontrol (методические рекомендации);

6 Создать шаблон текстового документа ms word, согласно методических рекомендаций, сохранить шаблон документа в одной директории с проектом под именем shablon.doc;

7 Для объекта button1 организовать событие onclick, внутри образовавшегося метода прописать программный код взаимодействия приложения delphi и mirosoft office (см. методические рекомендации);

8 Сохранить проект на диске в директории lab7_1 (**в той же папке, что и шаблон документа word**);

9 Выполнить отладку и компиляцию проекта;

10 Спроектировать вкладку «сведения о заказе», согласно рисунка 12;

11 Самостоятельно добавить в имеющееся событие по кнопке «оформить» программный код занесения данных о товаре в документ;

12 Сохранить проект, выполнить отладку и компиляцию проекта;

13 Оформить отчет о проделанной работе;

Задание повышенного уровня:

1 Выполнить индивидуальное задание согласно выданного варианта;

2 В каждом варианте предусмотреть создание таблицы в microsoft word (методические рекомендации);

3 Сохранить проект на диске в директории lab7_2;

4 Выполнить отладку и компиляцию проекта;

5 Оформить отчет о проделанной работе;

6 Защитить работу.

2.3 Содержание отчета

1 Цель, ход работы;

2 Постановка задачи №1, листинг программного модуля, результат работы приложения (скриншот формы проекта и документа word);

3 Формулировка индивидуального задания;

4 Листинг программного модуля индивидуального задания, результат работы приложения (скриншот формы проекта и документа word);

5 Вывод.

2.4 Контрольные вопросы

1 Охарактеризуйте основные понятия объектно-ориентированного программирования, опишите программное обращение;

2 Охарактеризуйте принципы объектно-ориентированного программирования, приведите примеры программного описания;

3 Какие виды методов можно организовать в классе, опишите их;

4 Как создать экземпляр сервера автоматизации, как обратиться к уже открытому экземпляру приложения?

5 Перечислите основные процедуры и функции, используемые для взаимодействия проекта delphi и microsoft word;

6 В чем особенность проектирования приложения и шаблона документа при работе с таблицей?

7 Охарактеризуйте процедуры, используемые для построения таблиц в mcrosoft word из проекта приложения delphi.

2.5 Методические рекомендации

Автоматизация - это протокол COM, который определяет, как одно приложение может получить доступ к объектам, находящимся в другом приложении или библиотеке DLL.

Сервер автоматизации - это приложение, которое предоставляет какие-либо услуги приложениям-клиентам. Примерами серверов автоматизации являются такие приложения, как Microsoft Word, Microsoft Excel, Microsoft Internet Explorer и др. Данные приложения могут контролироваться из приложений Delphi или других приложений. Для успешной работы с сервером автоматизации разработчику необходимо знать свойства и методы, которые предоставляют объекты сервера автоматизации. Описания свойств и методов можно получить из руководств разработчика по конкретным приложениям-серверам.

Диспетчер автоматизации (контроллер автоматизации) - это приложение-клиент, которое управляет сервером автоматизации при помощи объектов, которые поддерживают интерфейс IDispatch.

2.5.1 Создание и использование экземпляров серверов автоматизации

Для создания сервера автоматизации используется функция CfeateOleObject, описанная в модуле Comobj следующим образом:

function CreateOleObject(const ClassName: string): IDispatch;

Функция выдает ссылку на интерфейс IDispatch объекта, зарегистрированного в реестре Windows под именем ClassName. Для определения названия класса следует изучить документацию к программному продукту, предоставляющему сервер автоматизации. Для приложения Microsoft Word таким именем является «Word.Application», а для Microsoft Excel – «Excel.Application». Аналогичные названия классов имеют и другие компоненты Microsoft Office.

Если сервер автоматизации уже запущен, то ссылку на него можно получить с помощью функции GetActiveOleObject:

function GetActiveOleObject(const ClassName: string): IDispatch;

Если при вызове метода GetActiveOleObject система не может обнаружить запущенную версию заданного сервера автоматизации, то будет возбуждена исключительная ситуация класса EOleError.

Ссылки, которые возвращают функции GreateOleObject и GetActiveOleObject, следует сохранять в переменных для дальнейшего доступа к созданному или полученному объекту. Несмотря на то, что тип ссылки определен как IDispatch, переменная, в которую эта ссылка сохраняется, должна иметь тип Variant. Это связано с тем, что из данной переменной будут вызываться методы сервера автоматизации, которые не описаны в интерфейсе IDispatch.

```
Var
Object: Variant;
Object := CreateOleObject('Word.Application');
```

Использование экземпляра сервера автоматизации, то есть вызов его методов, осуществляется с помощью конструкций, обычных для вызова методов в Delphi:

<Ссылка на сервер>.<Название метода>(<Список параметров>);

Однако механизм, используемый для реального вызова, существенно отличается от вызова методов Delphi-классов. Название метода и список его параметров запаковываются в специальную структуру, которая затем передается методу invoke COM-объекта через ссылку, полученную при вызове функции CreateOleObject. Метод invoke определяет, какой именно его метод должен быть вызван, выполняет его, запаковывает результат и возвращает его в вызвавшую программу.

Объекты автоматизации поддерживают также и доступ к свойствам через специальным образом описанные методы.

Для разрушения структур данных, связанных с использованием COM-объекта в программе, следует присвоить ссылке на него значение Unassigned. Данная операция не закрывает запущенный сервер автоматизации:

```
Var
Object: Variant;
...
Object := CreateOleObject('Word.Application');
Object := Unassigned; // Разрушение программных структур
```

Рисунок 4 – Фрагмент структуры объекта Microsoft Word Object

2.5.2 Экспорт информации в Microsoft Word

Приложения, входящие в состав Microsoft Office, имеют иерархическую объектную структуру. Структура каждого приложения сложна и существенно отличается от структуры других приложений, что обусловлено различной направленностью самих приложений. Объекты иерархий Microsoft Office содержат в себе методы, количество которых приблизительно равно количеству действий, поддерживаемых при редактировании документов, поэтому их число очень велико. Иерархическая структура объектов отражает представление данных, редактируемых в том или ином приложении. Структура Microsoft Word включает в себя объект верхнего уровня Application, управляющий непосредственно приложением, и вложенные в него объекты типа Document, предоставляющие доступ к документам, открытым в данный момент. При добавлении очередного объекта Document сторонним приложением с помощью методов сервера автоматизации Microsoft Word открывает очередной документ. Аналогично объекты типа Document могут содержать в себе объекты типа Paragraph, представляющие собой ссылки на абзацы текста. Доступ к объектам различного уровня из программы-клиента, реализованной, например, на Delphi, осуществляется единообразно, через ссылки на вышестоящие в иерархии объекты. Объекты одного уровня и назначения, например, объекты типа Document, объединяются в одно свойство, так называемое семейство, которое имеет название типа объектов во множественном числе. Таким образом, объекты типа Document объединяются в свойство Documents.

Такие свойства являются, фактически списками, для которых можно определить количество элементов в каждом из них (свойство Count) и получить доступ к элементу с помощью функции Item, получающей в качестве параметра номер объекта в списке. Элементы нумеруются от единицы. Для передачи параметров в методы объектов автоматизации и присвоения значений их свойствам используется специальный тип данных OleVariant, основное отличие которого от типа Variant состоит в его совместимости с операционной системой, которая и поддерживает технологию COM.

Итак, для экспорта информации в Microsoft Word с использованием сервера автоматизации следует:

1 Создать экземпляр сервера автоматизации «Word.Application» (запустить Word) или получить ссылку на уже запущенный экземпляр;

2 Создать новый документ;

3 Вывести информацию в созданный документ;

4 Разрушить структуры, связанные с экземпляром сервера автоматизации в программе, а также ссылки на элементы его объектной иерархии.

2.5.2.1 Запуск сервера

Создание или получение ссылки на экземпляр сервера автоматизации выполняется с помощью функций CreateOleObject или GetActiveOleObject соответственно. В некоторых случаях целесообразно сначала пытаться подключиться к существующему серверу, а в случае неудачи запустить собственную версию.

2.5.2.2 Взаимодействие с сервером на уровне документа

Для создания нового документа следует добавить элемент в семейство Documents объекта «Word.Application» с помощью функции Add, которая вернет ссылку на созданный документ:

Add(Template: String, NewTemplate: Boolean): Document;

Строковый параметр Template определяет, на основе какого шаблона должен быть создан новый документ, а параметр NewTemplate указывает на то, что создаваемый документ сам должен являться шаблоном. Если метод вызывается без параметров, то новый документ создается на основе шаблона Normal (обычный) и является обычным документом.

Ссылку на вновь созданный документ, возвращаемую методом Add, следует

сохранить в переменной типа Variant для дальнейшего доступа к документу с целью вывода информации в него. При необходимости вывода информации в сложные формы новый документ можно создать на основе некоторого «шаблона» - ранее созданного и сохраненного документа. Параметры могут передаваться методу Add в обычной форме.

Закрыть документ после окончания вывода информации в него можно с помощью метода close.

2.5.2.3 Непосредственный вывод информации

Вывод информации в Microsoft Word аналогичен работе пользователя в редакторе и некоторым образом эмулирует ее. Так, с помощью объекта Selection, отражающего текущее выделение в документе, поддерживаются команды ввода текста и настройки его параметров. Если явного выделения не присутствует, то объект Selection отражает местонахождение текстового курсора. Заметим, что объект Selection принадлежит объекту «Word.Application», а не объекту Document.

Для вывода информации в объект Selection используется его метод TypeText. Для ввода символа перевода строки можно воспользоваться методом TypeParagraph объекта Selection. При последовательном выводе информации изменение автоматически установленного выделения обычно не требуется, однако, если возникнет такая необходимость, можно установить параметры выделения с помощью методов Move, MoveRight (сместить выделение вправо) и MoveLeft (сместить выделение влево):

Move(Unit: Integer, Count: Integer);

MoveRight(Unit: Integer, -Count: Integer, Extend: Boolean = False);

MoveLeft(Unit: Integer, Count: Integer, Extend: Boolean = False);

Методы сдвигают выделение на заданное параметром Count количество единиц. Значение параметра может быть положительным, либо отрицательным. В случае метода Move знак параметра Count определяет направление смещения

выделения. Отрицательное значение параметра указывает на смещение влево, а положительное – вправо. Методы MoveRight и MoveLeft изначально настроены на смещение выделения в заданную сторону (вправо и влево соответственно), поэтому отрицательное значение параметра Count в их вызове просто меняет направление смещения выделения. При использовании методов Move, MoveLeft и MoveRight, выделение, если оно существовало до их вызова, снимается. Этого можно избежать в функциях MoveLeft и MoveRight, если в качестве значения необязательного параметра Extend задать значение True (по умолчанию устанавливается False).

Параметр Unit определяет единицу смещения выделения. Некоторые его значения, указаны в таблице 4.

Таблица 4 – Некоторые значения параметра Unit

Значение	Единица смещения выделения
1	Один символ
2	Одно слово
3	Одно предложение
4	Один абзац
5	Одна строка
9	Один столбец таблицы, если выделение находится в таблице
10	Одна строка таблицы, если выделение находится в таблице
12	Одна ячейка таблицы, если выделение находится в таблице

Чтобы просто сбросить выделение, не изменяя его начального положения, можно воспользоваться методом Collapse объекта Selection.

2.5.2.4 Форматирование текстовой информации

Для форматирования текущего выделения, через объект Selection можно получить доступ к объекту Font, определяющему характеристики шрифта данного выделения. Основные свойства объекта Font перечислены в таблице 5.

Таблица 5 – Основные свойства объекта Font

Название свойства	Тип	Описание
Name	String	Название шрифта
Size	Integer	Размер шрифта
Bold	Boolean	Наличие атрибута «Полужирный»
Italic	Boolean	Наличие атрибута «Наклонный»
StrikeThrough	Boolean	Наличие атрибута «Перечеркнутый»
Subscript	Boolean	Символы в режиме «Нижний индекс»
Superscript	Boolean	Символы в режиме «Верхний индекс»
SmallCaps	Boolean	Все символы строчные
AllCaps	Boolean	Все символы заглавные

Документ, с точки зрения текстовой информации, состоит из набора (семейства) абзацев, представленных объектами Paragraph, доступ к каждому из которых возможен через функцию item объекта-семейства Paragraphs. Форматирование параграфа, редактирование которого производится в данный момент, осуществляется через свойство ParagraphFormat объекта Selection, а для объектов типа Paragraph возможности форматирования доступны напрямую. Доступные для изменения настройки абзаца включают выравнивание, наличие буквицы (первой буквы абзаца специального начертания), отступы первой строки от границы абзаца и отступы самой границы абзаца от краев страницы, название стиля абзаца, и множество других параметров, используемых в Word. Две часто используемых настройки абзаца – отступы абзаца и его выравнивание.

Отступы абзаца задаются свойствами Leftindent (отступ слева), Rightlndent (отступ справа) и FirstLinelndent (отступ первой строки от левой границы абзаца) объекта ParagraphFormat. Значения отступов задаются вещественными числами в

условных единицах, которые можно получить из сантиметров или дюймов с помощью методов объекта «Word.Application» CentimetersToPoints и InchesToPoints.

Выравнивание редактируемого (текущего) абзаца выполняется с помощью свойства Alignment объекта ParagraphFormat. Выравнивание всех абзацев документа можно выполнить через одноименное свойство объекта-семейства Paragraphs. В качестве значений, определяющих выравнивание, могут использоваться: 0 (выравнивание по левому краю), 1 (выравнивание по центру), 2 (выравнивание по правому краю) и 3 (выравнивание по ширине).

Для установки одинакового выравнивания для всех абзацев можно воспользоваться объектом-семейством Paragraph.

2.5.2.5 Использование закладок

Microsoft Word поддерживает возможность работы с закладками – неотображаемыми атрибутами документа, управление которыми (добавление, удаление и переход на закладку) осуществляется с помощью диалога пункта главного меню Вставка→Закладка. Приложение, которое является OLE-клиентом, может обратиться к семейству Bookmarks закладок для доступа к каждой из них, или к объекту Selection для перехода (перемещения выделения) на закладку, заданную именем.

Для перехода на закладку следует использовать метод Goto объекта Selection.

Selection. GoTo (What: Integer; Name: String);

Параметр What указывает тип элемента, на который следует переместиться. Параметр Name задает название закладки, указанное при ее добавлении в документ.

Использование закладок существенно упрощает подготовку унифицированных документов, например, анкет, в которых большая часть информации является вспомогательной, а на ее основе следует заполнить какие-либо поля данных. Места, куда должна быть введена информация, можно пометить закладками, по которым Delphi-программа будет перемещать выделение с целью вывода информации

методом TypeText объекта Selection.

2.5.2.6 Управление приложением Microsoft Word

Приложение Microsoft Word, которое является сервером автоматизации, может присутствовать на экране в момент обращения к нему клиента, а может быть, скрыто. Видимость приложения определяется логическим свойством Visible. Управление видимостью приложения может быть необходимо, чтобы пользователь не смог помешать процессу экспорта информации. Приложения Office устроены таким образом, что им все рано, кто вводит информацию – стороннее приложение через сервер автоматизации или пользователь с помощью интерфейса. Таким образом, если Delphi-приложение осуществляет длительный экспорт информации, используя метод TypeText объекта Selection, пользователь имеет возможность переключиться в окно Word и, например, изменить положение текстового курсора. В результате таких действий изменится состояние объекта Selection, и информация будет выведена не по порядку, а из того места, которое указал пользователь. Для того, чтобы запретить пользователю изменять выделение во время процесса экспорта информации, можно скрыть окно приложения с экрана на это время.

Еще одна интересная особенность использования сервера автоматизации Microsoft Word вытекает из принадлежности объекта Selection к объекту «Word.Application», а не к объекту Document, с которым работает программа. Если ссылка на сервер автоматизации не создана в программе функцией CreateOleObject, а получена из функции GetActiveOleObject, то переключение пользователем в другое окно приложения Word, используемого программой, также приведет к изменению смысла свойства Selection. После переключения в другой документ данное свойство будет определять выделение в этом документе, что является недопустимым, так как в него будет осуществляться вывод информации. Для избежание таких проблем не следует использовать сервера автоматизации, полученные функцией GetActiveOleObject, если вывод информации может занять длительное время. Заметим, что даже вывод нескольких строк может дать пользователю возможность

переключения между приложениями, поэтому более правильно создавать новый сервер автоматизации при использовании методов объекта «Word.Application» вообще.

2.5.3 Проектирование приложения

Для организации взаимодействия приложения Delphi с MS Word необходимо организовать форму следующего вида

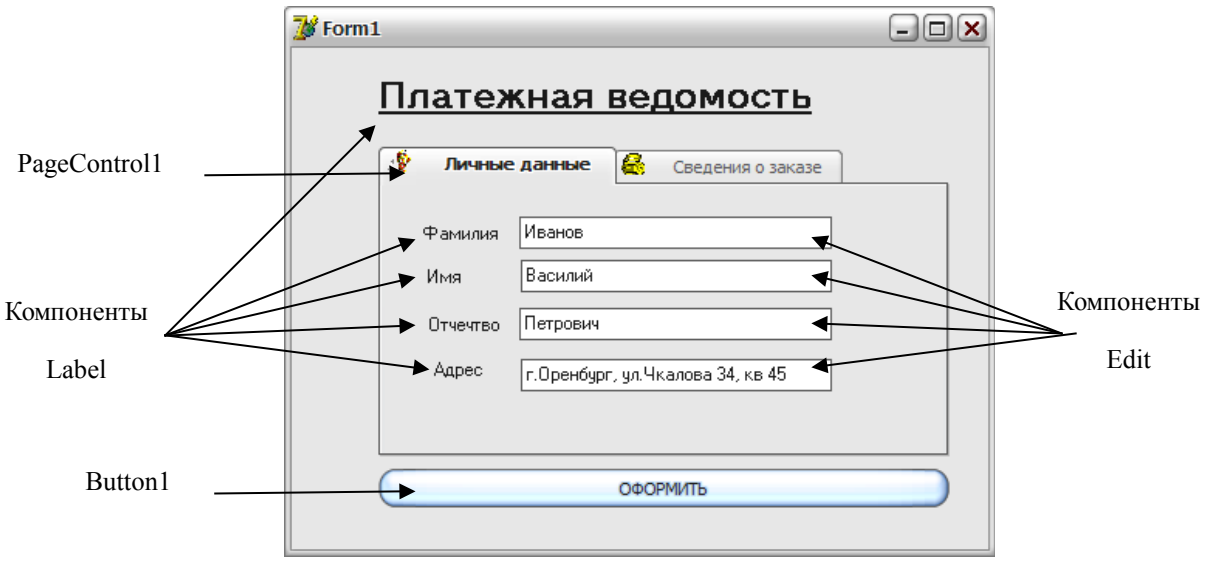
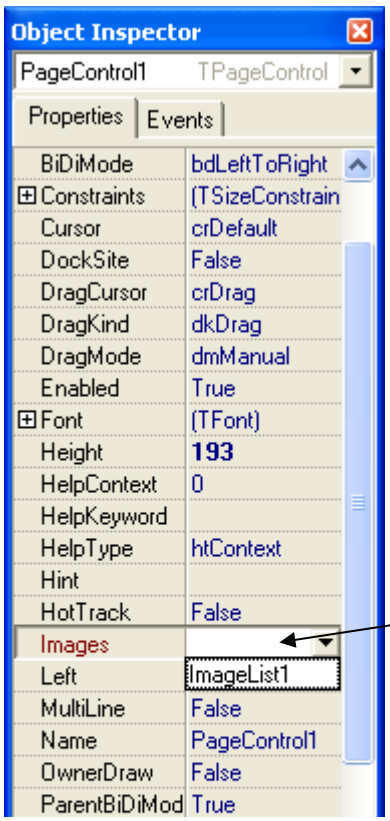


Рисунок 5 – Рекомендуемый внешний вид

Для того, чтобы на вкладках PageControl разместить иконки, необходимо на форму поместить дополнительный компонент ImageList1  .

Используя свойство Images компонета PageControl, соединяем его с ImageList1 (рисунок 6).



свойство Images

компонента PageControl

Рисунок 6 – Свойства PageControl

Двойной щелчок по компоненту ImageList1 открывает дополнительное окно, используя которое можно загрузить иконки для вкладок (рисунок 7).

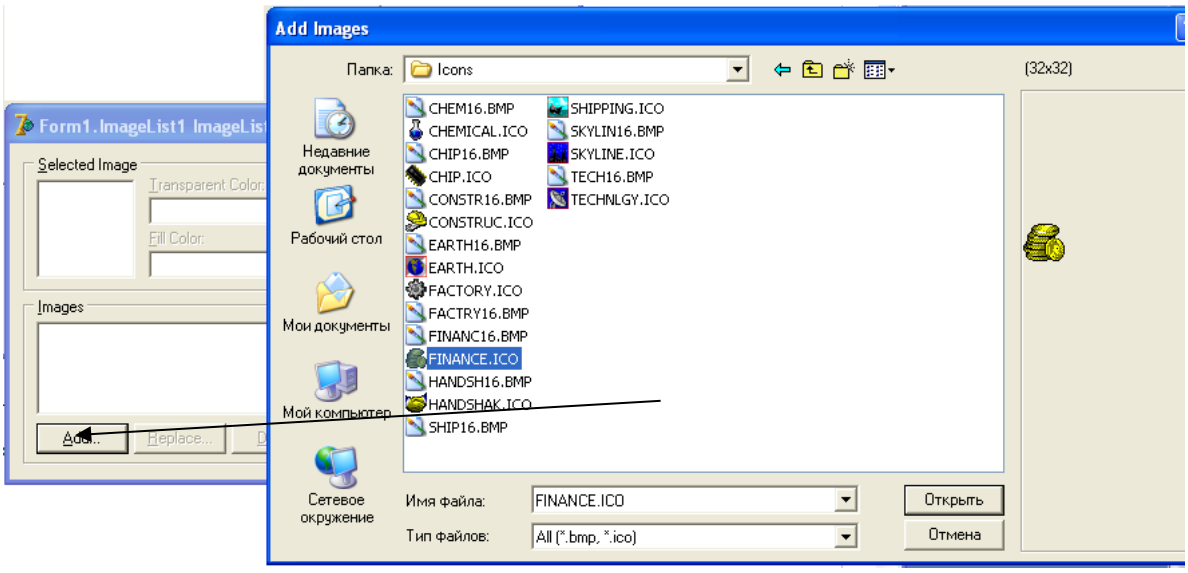


Рисунок 7 – Добавление иконок в ImageList

Для каждой вкладки PageControl, настраивается свойство ImageIndex, определяющее номер иконки в компоненте ImageList.

Для использования серверов автоматизации в разделе uses необходимо подключить модуль **COMOBJ**.

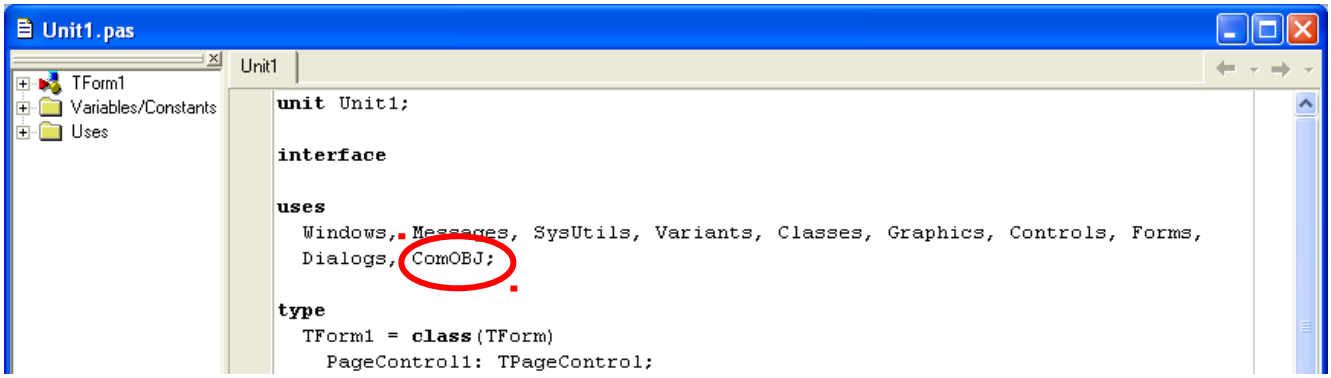


Рисунок 8 – Подключение дополнительного модуля

Создайте шаблон документа Word на основе приведенного ниже:

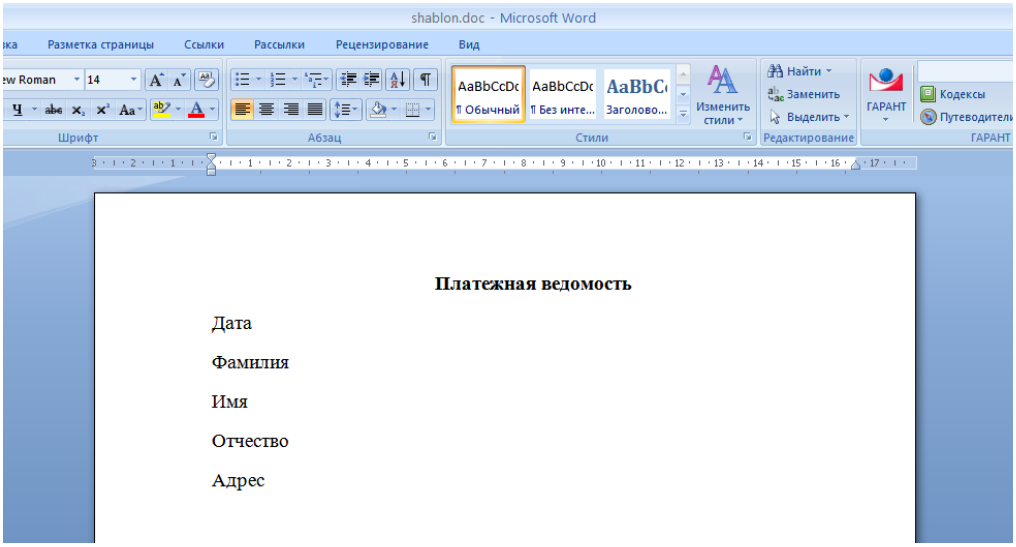


Рисунок 9 – Шаблон документа Word (shablon.doc)

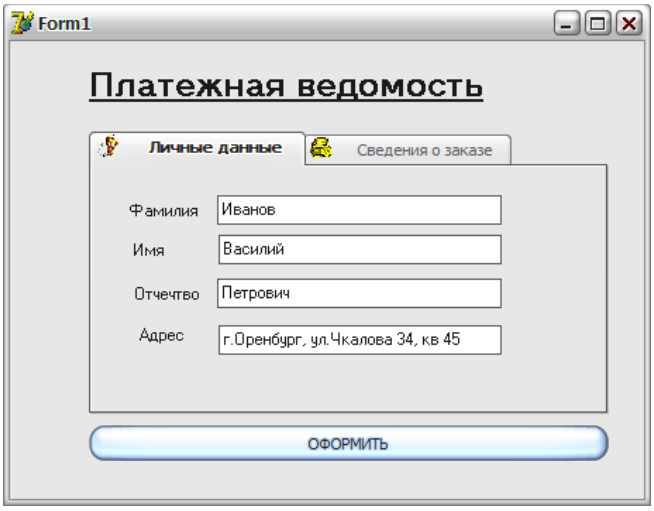


Рисунок 10 – Главная форма приложения

Примечание: Путь к шаблону документа необходимо прописать через процедуру *ExtractFilePath(<имя файла>)*, которая автоматически определяет местоположение файла, указанного в параметрах процедуры. Параметром в нашем случае является исполнимый файл приложения *Application.ExeName*.

if fileExists(ExtractFilePath(Application.ExeName)+'shablon.doc') then…..

Обработчик события OnClick компонента Button1:

```
procedure TForm1.Button1Click(Sender: TObject);
var
docum,wd,vend,vstart,a,b:OleVariant;
j,ilengy : integer;
begin
if fileExists('shablon.doc') then
begin
wd:=createOleObject('Word.application');
docum:=wd.Documents.Open('shablon.doc');
ilengy:=Length(docum.range.text);
for j:=0 to ilengy-5 do
begin
a:=j;
```

```
b:=j+4;
if docum.Range(a,b).text='Дата' then begin
vstart:=j;
vend:=j+4;
end;
end;
docum.Range(vstart,vend).Select;
wd.Selection.InsertAfter(': '+DateToStr(date));
docum.Range(vstart,vend).Select;
Wd.Selection.Font.Bold:=1;
Wd.Selection.Font.Size:=16;
wd.Selection.Font.color:=clblack;
ilengy:=Length(docum.range.text);
for j:=0 to ilengy-8 do begin
a:=j;
b:=j+7;
if docum.Range(a,b).text='Фамилия' then begin
vstart:=j;
vend:=j+7;
end;
end;
docum.Range(vstart,vend).Select;
wd.Selection.InsertAfter(': '+Edit1.Text);
docum.Range(vstart,vend).Select;
Wd.Selection.Font.Bold:=1;
Wd.Selection.Font.Size:=16;
wd.Selection.Font.color:=clblack;
ilengy:=Length(docum.range.text);
for j:=0 to ilengy-4 do begin
a:=j;
```

```
b:=j+3;
if docum.Range(a,b).text='Имя' then begin
vstart:=j;
vend:=j+3;
end;
end;
docum.Range(vstart,vend).Select;
wd.Selection.InsertAfter(': '+Edit2.Text);
docum.Range(vstart,vend).Select;
Wd.Selection.Font.Bold:=1;
Wd.Selection.Font.Size:=16;
wd.Selection.Font.color:=clblack;
ilengy:=Length(docum.range.text);
for j:=0 to ilengy-9 do begin
a:=j;
b:=j+8;
if docum.Range(a,b).text='Отчество' then begin
vstart:=j;
vend:=j+8;
end;
end;
docum.Range(vstart,vend).Select;
wd.Selection.InsertAfter(': '+Edit3.Text);
docum.Range(vstart,vend).Select;
Wd.Selection.Font.Bold:=1;
Wd.Selection.Font.Size:=16;
wd.Selection.Font.color:=clblack;
ilengy:=Length(docum.range.text);
for j:=0 to ilengy-6 do begin
a:=j;
```

```
b:=j+5;
if docum.Range(a,b).text='Адрес' then begin
vstart:=j;
vend:=j+5;
end;
end;
docum.Range(vstart,vend).Select;
wd.Selection.InsertAfter(': '+Edit4.Text);
docum.Range(vstart,vend).Select;
Wd.Selection.Font.Bold:=1;
Wd.Selection.Font.Size:=16;
wd.Selection.Font.color:=clblack;
wd.visible:=true;
end;
end;
end.
```

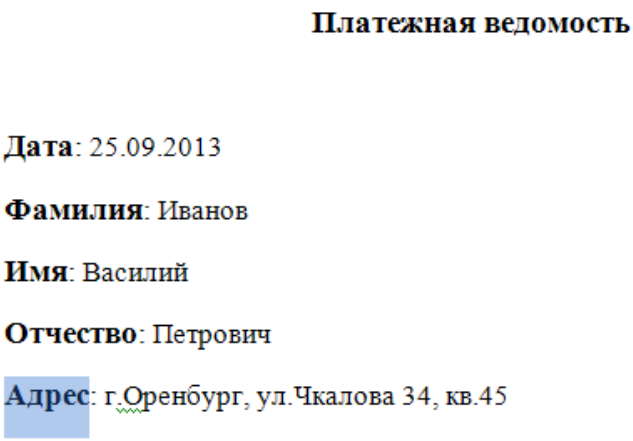


Рисунок 11 – Результат работы проекта

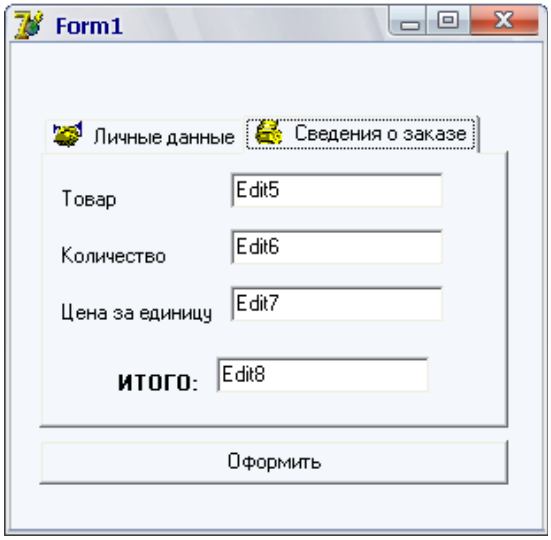


Рисунок 12 – Вкладка «Сведения о заказе»

2.5.4 Работа с таблицами

Документ Word может создавать таблицы, которые как объекты объединены в коллекцию Tables.

Свойство Count:integer определяет количество таблиц, используемых в документе.

Метод Add добавляет таблицу в документ. При создании таблицы определяется область, где будет создана таблица, и ее основные параметры – количество столбцов и строк.

Например,

W.ActiveDocument.Tables.Add(range:=W.ActiveDocument.Range, NumRows:=2, NumColumns:=3);

Данная процедура создаст таблицу непосредственно в документе. Первый аргумент метода определяет область, где будет создана таблица, - весь документ. Следующие аргументы определяют количество строк и столбцов.

Создание таблицы в конце документа:

Var

docum: variant;

Begin

docum.range(Wd.ActiveDocument.Range.End-1, Wd.ActiveDocument.Range.End-1).select;

Wd.ActiveDocument.Tables.Add(wd.selection.range,2,3);

….

End;

Для того, чтобы осуществить добавление текста в таблицу удобно воспользоваться методом MoveRight (описание параметров см.выше), который позволит перемещаться по ячейкам таблицы.

wd.Selection.InsertAfter('Наименование товара');

Wd.Selection.MoveRight(12, 1, EmptyParam);

wd.Selection.InsertAfter('Итого');

Wd.Selection.MoveRight(12, 1, EmptyParam);

……

2.6 Индивидуальные задания

1 Создать проект осуществляющий формирование отчета об оборудовании кабинета. Примерный вид отчета приведен на рисунке 13;

Сведения о кабинете

Номер…..

Ответственный _____

ФИО

Наименование кабинета_____

Оборудование

№ п/п	Наименование	Количество	Инвентарный номер

Рисунок 13 – Примерный вид отчета

2 Создать проект осуществляющий формирование списка студентов учебной группы. Отчет содержит данные о номере группы, кураторе, список

студентов с указанием их среднего балла за месяц;

3 Создать проект осуществляющий формирование списка товара на складе. Отчет содержит сведения о номере склада, адресе склада, заведующем и перечень товара с указанием цены и количества;

4 Создать проект осуществляющий формирование списка домов в заданном районе. Отчет содержит сведения о дате формирования списка, наименовании района, перечень адресов с указанием количества жителей в доме;

5 Создать проект осуществляющий формирование списка призывников в определенном районе города. Отчет содержит сведения о дате формирования списка, наименование района, дата призыва, перечень призывников с указанием фамилии, инициалов, дате рождения;

6 Создать проект осуществляющий формирование отчета о новорожденных на определенную дату. Отчет содержит сведения о наименовании роддома, фио заведующего, дата составления списка, перечень новорожденных с указанием фамилии матери, пола, веса и роста;

7 Создать проект осуществляющий формирование списка домов, входящих в состав тсж. Отчет содержит сведения о наименовании тсж, фио управляющего, адрес тсж, перечень домов (улица, номер дома);

8 Создать проект осуществляющий формирование списка произведений определенного автора. Отчет содержит сведения об авторе (фамилия, имя, отчество, годы жизни), перечень произведений (наименование, год издания, жанр);

9 Создать проект осуществляющий формирование списка детей в группе детского сада. Отчет содержит данные о номере группы, воспитателях, перечень детей с указанием фамилии, имени, даты рождения;

10 Создать проект осуществляющий формирование отчета о занятости в кружках и секция учеников определенного класса. Отчет содержит данные о номере класса, классном руководителе, перечень учеников с указанием фамилии, имени, наименование кружка, количество посещений в неделю;

11 Создать проект осуществляющий формирование списка дисциплин, читаемых на определенном курсе заданной специальности. Отчет содержит данные о заведующем отделением, наименовании специальности, номере курса, перечень дисциплин с указанием наименования дисциплины, количество лекционных часов, количество лабораторных занятий;

12 Создать проект осуществляющий формирование рецепта для больного. Отчет содержит сведения о враче, дате обследования, фио пациента, перечень лекарств с указанием наименования лекарства, количество таблеток в день, курс лечения (в днях);

13 Создать проект осуществляющий формирование списка городов определенной области РФ. Отчет содержит сведения о наименовании области, наличие внешних границ с другими государствами, перечень городов с указанием наименования города и численности населения;

14 Создать проект осуществляющий формирование рецепта приготовления блюда. Отчет содержит сведения о полном наименовании блюда, общее время приготовления, количество калорий, перечень ингредиентов с указанием наименования ингредиента, количество в граммах, примерная цена;

15 Создать проект осуществляющий формирование списка участков, закрепленных за поликлиникой. Отчет содержит сведения о номере поликлинике, фио заведующего, адрес, перечень участков с указанием номера участка, участковом враче, адреса домов, на данном участке.

3 Лабораторная работа №8. Подключение базы данных к приложению Delphi. Работа с технологией ADO

3.1 Цель работы

Научиться подключать базу данных к приложению Delphi. Изучить основные принципы работы технологии ADO.

3.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Создать в microsoft access базу данных *kadri.mdb* для отдела кадров предприятия. Поля для создаваемой таблицы *sotrudnik*: код сотрудника (*kod*), фамилия (*fam*), имя (*name*), отчество (*otch*), пол (*pol*), дата рождения (*rozhd*), образование (*obraz*), домашний адрес (*adres*), телефон (*tel*);

3 Создать новый проект delphi, осуществляющий подключение базы данных с помощью технологии ado;

4 Спроектировать форму приложения, согласно рисунка 16, представленного в методических рекомендациях; поместить на форму компонент *adoconnection1;* обеспечить связь с базой данных при помощи механизма *ado;*

5 Поместить на форму компоненты *adotable1, datasource1*, *dbgrid1, dbnavigator1*. Связать данные компоненты между собой (методические рекомендации);

6 Организовать поиск записей в таблице по полю *фамилия* с помощью метода *locate* (методические рекомендации *п. 3.5.2*);

7 Организовать фильтрацию записей в таблице по полю *образование* с помощью метода *filter* и *filtered* (методические рекомендации *п. 3.5.3*);

8 Организовать сортировку записей по выбранному столбцу с помощью процедуры *sort* (методические рекомендации *п. 3.5.4*);

9 Провести отладку и компиляцию проекта;

10 Сохранить проект на диске в директории lab8_1;

11 Оформить отчет о проделанной работе;

Задание повышенного уровня:

12 Выполнить индивидуальное задание согласно выданного варианта;

13 В проекте осуществить подключение базы данных с помощью технологии ado, организовать поиск, фильтрацию данных по заданным полям;

14 Сохранить проект на диске в директории lab8_2;

15 Выполнить отладку и компиляцию проекта;

16 Оформить отчет о проделанной работе;

17 Защитить работу.

3.3 Содержание отчета

1 Цель, ход работы;

2 Постановка задачи, листинг программного модуля, результат работы приложения;

3 Формулировка индивидуального задания;

4 Листинг программного модуля индивидуального задания, результат работы приложения;

5 Вывод.

3.4 Контрольные вопросы

1 Охарактеризуйте основные понятия объектно-ориентированного программирования;

2 Охарактеризуйте принципы объектно-ориентированного программирования (полиморфизм, инкапсуляция, наследование);

3 Привести примеры программного описания принципов объектно-ориентированного программирования;

4 В чем особенность использования технологии ado?

5 Как осуществить подключение базы данных к приложению, используя компонент *adoconnection*?

6 Как организовать фильтрацию записей в таблице по определенному полю?

7 Как организовать поиск записей в таблице по определенному полю?

3.5 Методические рекомендации

3.5.1 Подключение базы данных к приложению с помощью технологии ADO

Наряду с традиционными инструментами доступа к данным Borland Database Engine и ODBC в приложениях Delphi можно применять технологию Microsoft ActiveX Data Objects (ADO), которая основана на возможностях COM, а именно интерфейсов OLE DB. По сути, ADO - это надстройка над технологией OLE DB, посредством которой можно связываться с различными данными приложений Microsoft.

Технология ADO завоевала популярность у разработчиков, благодаря универсальности — базовый набор интерфейсов OLE DB имеется в каждой современной операционной системе Microsoft. Поэтому для обеспечения доступа приложения к данным достаточно лишь правильно указать провайдер соединения

ADO и затем переносить программу на любой компьютер, где имеется требуемая база данных и, конечно, установленная ADO.

В палитре компонентов Delphi есть страница ADO, содержащая набор компонентов, позволяющих создавать полноценные приложения БД, обращающиеся к данным через ADO.

Основные компоненты вкладки ADO

– *TADOConnection* используется для указания базы данных и работы транзакциями;

– *TADOTable* – таблица доступная через ADO;

– *TADOQuery* – запрос к базе данных. Это может быть как запрос, в результате которого возвращаются данные и базы (например, *SELECT*), так и запрос, не возвращающий данных (например, *INSERT*);

– *TADOStoredProc* – вызов хранимой процедуры. В отличие от BDE и InterBase хранимые процедуры в ADO могут возвращать набор данных, поэтому компонент данного типа является потомком от TDataSet, и может выступать источником данных в компонентах типа TDataSource;

– *TADOCommand* и *TADODataSet* являются наиболее общими компонентами для работы с ADO, но и наиболее сложными в работе. Оба компонента позволяют выполнять команды на языке провайдера данных (так в ADO называется драйвер базы данных).

Все компоненты должны связываться с базой данных. Делается это двумя способами либо через компонент *TADOConnection* либо прямым указанием базы данных в остальных компонентах. К *TADOConnection* остальные компоненты привязываются с помощью свойства *Connection*, к базе данных напрямую через свойство *ConnectionString*.

Для примера создадим базу данных *kadri.mdb* в Microsoft Access для отдела кадров предприятия. Поля для создаваемой таблицы *Sotrudnik*: код сотрудника (*Kod*), фамилия (*Fam*), имя (*Name*), отчество (*Otch*), пол (*Pol*), дата рождения (*Rozhd*), образование (*Obraz*), домашний адрес (*Adres*), телефон (*Tel*).

Загружаем *Delphi*, создаем новый проект. Добавляем в модуль компонент

ADOConnection1  с вкладки *ADO* палитры компонентов. Этот компонент обеспечит связь других компонентов с базой данных при помощи механизма *ADO*. Связь обеспечивается свойством компонента *ConnectionString*.

Щелкнем дважды по свойству *ConnectionString* компонента *ADOConnection*. Откроется окно подключения компонента к ADO:

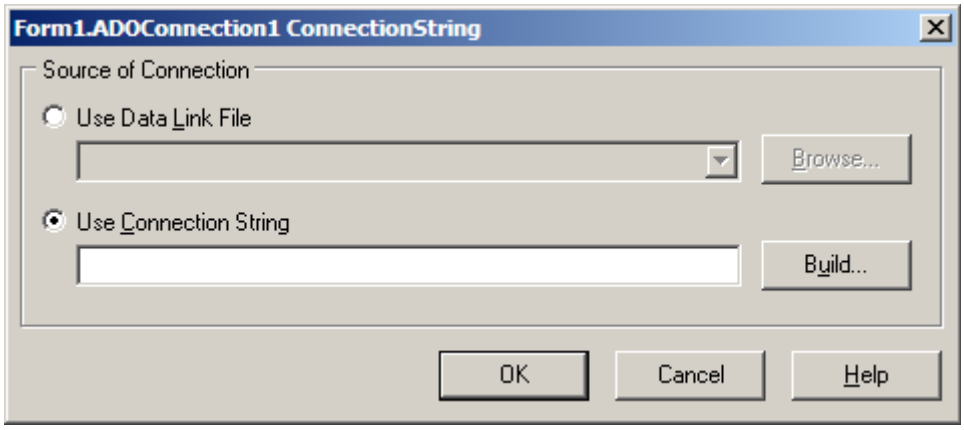


Рисунок 14 – Окно подключения компонента к ADO

Выберем переключатель *Use Connection String*. Нажмем кнопку *Build*. Открывается новое окно, содержащее настройки подключения.

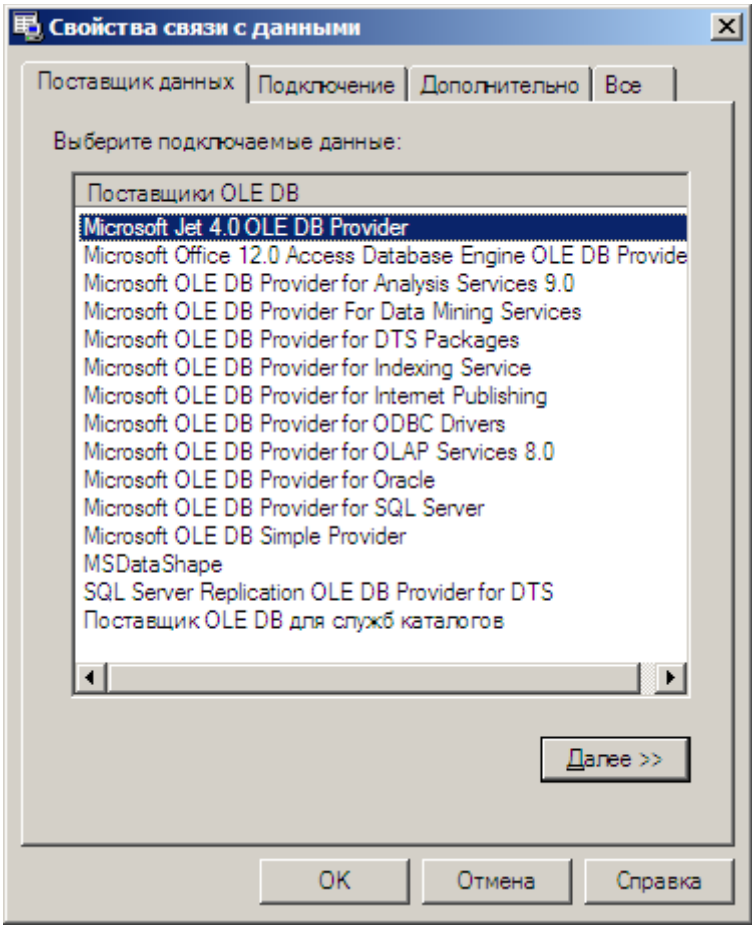


Рисунок 15 – Окно настроек подключения

Предлагается выбрать поставщика OLE DB, или иначе, указать нужный для подключения драйвер. Для связи с базой данных *MS Access* больше всего подходит *" Microsoft Jet 4.0 OLE DB Provider "*. Jet - это название механизма работы с СУБД, встроенного в MS Access. Этот механизм поддерживает как собственные БД MS Access, имеющие расширение **.mdb*, так и ODBC. Его и выделяем в списке.

Нажимаем на кнопку *Далее*, либо переходим к вкладке *Подключение*. Здесь нам нужно выбрать или ввести базу данных. Если мы выберем базу данных, то есть, нажмем на кнопку с тремя точками, откроем диалог выбора и найдем там наш файл, то база данных будет привязана к указанному адресу. Однако, в случае, если вы поместили файл с базой данных (в нашем случае *kadri.mdb*) там же, где находится программа, и не желаем зависеть от определенной папки (ведь пользователь может переместить нашу программу), то нужно вручную вписать только имя файла с БД,

без всякого адреса. В этом случае мы не сможем проверить подключение, нажав на кнопку *Проверить подключение*. Укажем только имя файла - *kadri.mdb*. Нажмем на кнопку *ОК*.

Закрываем окно *редактора связей*. Однако перед этим переведем свойство *LoginPrompt* компонента *ADOConnection* в *False*. Если этого не сделать, то при каждой попытке соединиться с базой данных будет выходить *запрос* на пользовательское имя и пароль, а подключаемая база данных без пароля. Теперь свойство *Connected* переведем в *True*. Если удалось это сделать, и не вышло никаких сообщений об ошибке, то подключение состоялось.

Установим на форму компонент *ADOTable*  с вкладки *ADO*. Компонент *ADOTable* предназначен для создания набора данных. В свойстве *Connection* указываем наш *ADOConnection1* (можно просто выбрать из выпадающего списка). В свойстве *TableName* выбираем таблицу, которую нам необходимо вывести.

Установим компонент *DataSource*  из вкладки *Data Access* палитры компонентов. Компонент *DataSource* предназначен для организации связи с наборами данных, и служит посредником между такими компонентами *ADOTable*, *ADOQuery* и между компонентами отображения данных *DBGrid*. Свойство *DataSet* этого компонента меняем на *ADOTable1*.

С вкладки *DataControls* поместим на форму компонент отображения данных *DBGrid*  (сетку, отображающую все данные набора данных в виде таблицы, и позволяющую редактировать их). Свойству *DataSource* присваиваем значение *DataSource1*. С этой же вкладки поместим компонент *DBNavigator*  , предназначенный для перемещения по записям набора данных, для вставки новой записи или удаления старой, для перевода набора данных в режим редактирования или для подтверждения сделанных изменений в наборе данных. Свойству *DataSource* присваиваем значение *DataSource1*.

Значение свойства *Active* компонента *ADOTable1* делаем равным *true*.

Внешний интерфейс приложения представлен на рисунке 16.

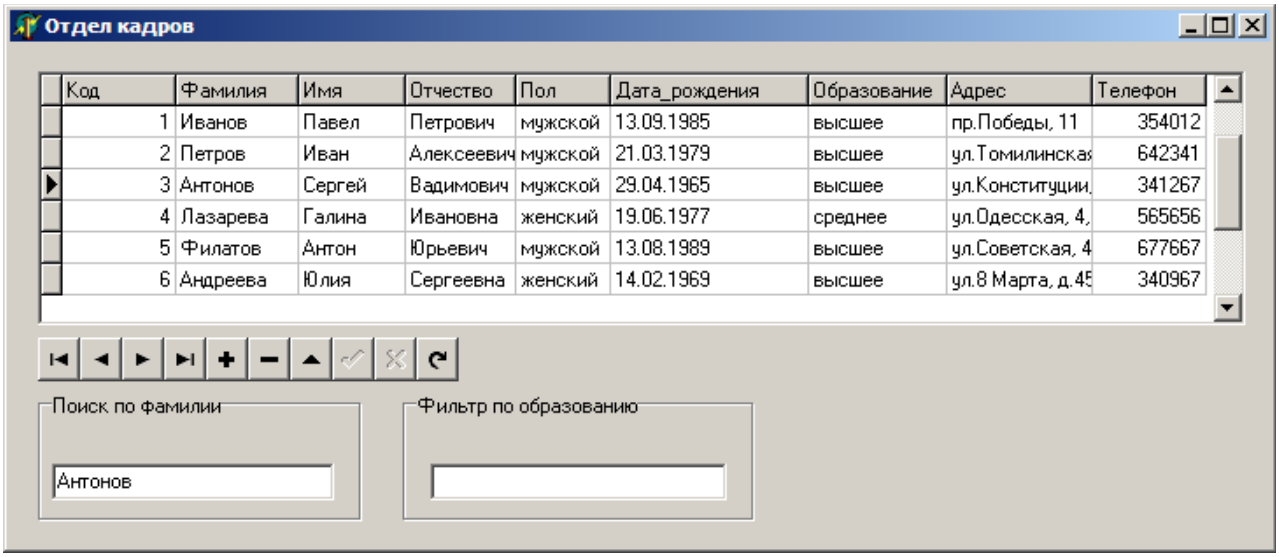


Рисунок 16 – Внешний интерфейс приложения

3.5.2 Организация поиска записей

Добавим компоненты на форму для организации поиска в таблице. Разместим на форме компонент *GroupBox1*, меняем свойство *Caption* на значение *Поиск по фамилии*. На компонент *GroupBox1* помещаем *Edit1*. Далее в обработчике события *OnChange* однострочного редактора *Edit1* организуем вызов метода *Locate*, который позволяет произвести поиск нужной записи. Код обработчика события приведен ниже:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  ADOTable1.Locate('Fam', Edit1.Text, [loCaseInsensitive,loPartialKey]);
end;
```

Разберем код более подробно. Строка *Table1.Locate* организует поиск записи в таблице *Sotrudnik*. Первый параметр этой функции - поля, значения которых нужно проверять. В данном случае мы ищем запись по одному полю *Fam* (*Фамилия*). Второй параметр – это шаблон поиска, вводимое в поле *Edit1* значение для поиска.

Третий параметр – опции поиска – имеет тип *TLocateOptions* и позволяет задавать набор из двух параметров поиска: *loCaseInsensitive* и *loPartialKey*.

Установка первого из них отменяет чувствительность к регистру в текстовых полях, а второй позволяет искать запись частично соответствующие заданному условию.

Функция *Locate* возвращает значение типа *boolean*, указывающее на успешность поиска.

3.5.3 Организация фильтрации записей

В отличие от методов поиска, предполагающих извлечение данных из хранилища данных, фильтрация предполагает отбор уже извлеченных данных в клиентском приложении. Для реализации данного подхода в *Delphi* в компонентах доступа к данным введены два свойства *Filter* и *Filtered*. Установка свойства *Filtered* типа *boolean* в *true* переводит компонент в режим фильтрации. В свойстве *Filter* при этом можно определить значение фильтра для отбора записей.

Параметры фильтрации можно задать с помощью свойств *FilterOptions* типа *TFilterOptions*. Это свойство принадлежит к множественному типу и может принимать комбинации двух значений:

- *foCaseIntensitive* - регистр букв не учитывается;
- *foNoPartialCompare* - выполняется проверка на полное соответствие содержимого поля и значение, заданного для поиска. Обычно применяется для строк. Если известны только первые символы или символ строки, то нужно указать их в выражении фильтра, заменив остальные символы символом "*" и выключив значение FoNoPartialCompare.

По умолчанию все фильтры выключены и свойство FilterOptions имеет значение [].

Помещаем на форму компонент *GroupBox2*, меняем свойство *Caption* на значение *Фильтр по образованию*. На компонент *GroupBox2* помещаем *Edit2*. Далее в обработчике события *OnChange* однострочного редактора *Edit2* организуем назначение свойств *Filter* и *Filtered*, которые позволяют произвести фильтрацию записей. Код обработчика события приведен ниже:

```
procedure TForm1.Edit2Change(Sender: TObject);
begin
  ADOTable1.FilterOptions:=[];
  ADOTable1.Filter:='Obraz='"+Edit2.Text+"";
  ADOTable1.Filtered:=True;
end;
```

3.5.4 Сортировка записей в таблице

В *TADOTable* есть метод *Sort*, который позволяет сортировать данные в таблице. В обработчике события *OnTitleClick* компонента *DBGrid1* организуем вызов метода *Sort*, который позволяет произвести сортировку записей по выбранному столбцу. Код обработчика события приведен ниже:

```
procedure TForm1.DBGrid1TitleClick(Column: TColumn);
begin
   ADOTable1.Sort:= Column.FieldName;
end;
```

Объект *Column* типа *TColumn* описывает отдельный столбец сетки *DBGrid*. Свойство *FieldName* возвращает имя выбранного столбца.

3.6 Индивидуальные задания

1 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Biblioteka*, содержащей информацию об имеющихся книгах: код книги, название, автор, издательство, год издания, количество страниц;

2 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Prepodavatel,* содержащей информацию о преподавателях

кафедры: код преподавателя, фамилия, имя, отчество, наименование кафедры, количество часов;

3 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Pacient*, содержащей информацию о пациентах больницы: код пациента, фамилия, имя, отчество, отделение больницы, номер палаты, диагноз;

4 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Sklad*, содержащей информацию о товаре, имеющемся на складе: код товара, наименование товара, единица измерения, количество товара на складе, дата поступления товара на склад;

5 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Ucheb_Gruppa*, содержащей информацию о группах колледжа: номер группы, год поступления, курс, количество студентов в группе, куратор;

6 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Marshrut*, содержащей информацию о маршрутных автобусах: номер маршрута, количество автобусов на маршруте, начальный пункт, конечный пункт, количество пассажиров в день;

7 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Airport*, содержащей информацию о рейсах: номер самолета, число мест, пункт вылета, пункт назначения, дата вылета, дата прилета;

8 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *PK*, содержащей информацию о персональных компьютерах организации: номер персонального компьютера, фирма-изготовитель, тактовая частота, объем ОЗУ, объем жесткого диска, дата выпуска ПК;

9 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Zoopark*, содержащей информацию животных зоопарка: код животного, название вида животного, континент обитания, суточное потребление корма, номер помещения, наличие водоема;

10 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *PP*, содержащей информацию программном обеспечении организации: код программного продукта, наименование программного продукта, версия, фирма, дата выпуска, прикладная область, стоимость лицензии;

11 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *PoleznIskop*, содержащей информацию о добыче полезных ископаемых: код полезного ископаемого, наименование полезного ископаемого, единица измерения, название месторождения, годовая добыча, себестоимость за единицу;

12 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Spectakl*, содержащей информацию о спектаклях театра: код спектакля, наименование спектакля, режиссер, дата проведения, количество билетов, цена билета;

13 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Apteka*, содержащей информацию о имеющемся лекарстве: код лекарства, наименование лекарства, производитель, тип, количество, цена;

14 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Menu*, содержащей информацию о блюдах: код блюда, наименование блюда, категория, калорийность, единица измерения, цена за единицу измерения;

15 Подключить базу данных к приложению Delphi с помощью технологии ADO; организовать поиск, фильтрацию, сортировку записей в таблице. Поля для создаваемой таблицы *Country*, содержащей информацию о странах: код страны, континент, столица, площадь территории, количество населения, язык.

Список использованных источников

1 Хомоненко А.Д. Delphi 7 / А.Д. Хомоненко [и др.]. – СПб.: БХВ-Петербург, 2008. – 1216 с. : ил.

2 Культин Н.Б. Delphi в задачах и примерах [Комплект] / Н.Б. Культин. – 3 изд. – СПб.: БХВ-Петербург, 2012. – 228 с. : ил. – 1 электрон. опт. диск (CD-ROM).

3 Катаев М.Ю. Объектно-ориентированное программирование : лабораторный практикум / М.Ю. Катаев. – Томск: Томский межвузовский центр дистанционного образования, 2007. – 68 с.

4 Марченко А.И. Программирование в среде TurboPascal 7.0 : учебное пособие / А.И. Марченко, Л.А. Марченко. – 9 изд. – СПб.: Корона-Век, 2007. – 458 с.

5 Бескоровайный И.В. Азбука Delphi : программирование с нуля / И.В. Бескоровайный. – Новосибирск: Сиб. унив. изд-во, 2008. – 112 с.

6 Фаронов В. В. Программирование баз данных в Delphi 7 : учебный курс / В.В. Фаронов. – СПб.: Питер, 2007. – 464 с. : ил.

7 Хомоненко А. Delphi 7. [Электронный ресурс] : Электронно-библиотечная система ibooks.ru. / А. Хомоненко, В. Гофман, Е. Мещеряков. – СПб.: Айбукс. – 2010. – Режим доступа : http://ibooks.ru/product.php?productid=18411

8 Профессиональные программы для разработчиков [Электронный ресурс] : Delphi World / под ред. Н. Акулова. – Алматы: WDS, 2002. – Режим доступа : http://delphiworld.narod.ru/.

9 Королевство Delphi. Виртуальный клуб программистов [Электронный ресурс] / под ред. Е. Филипповой. – М.: DOTNETPARK, 1998. – Режим доступа : http://delphikingdom.ru/index.asp.

10 Высокоуровневые методы математики и информатики [Электронный ресурс] : Киберфак / под ред. Ni-Cd. – М.: МИФИ, 2007-2014. – Режим доступа : http://cyberfac.ru/publ/informatika/vysokourovnevye_metody_informatiki_i_programmirovanija/konstruktor_i_destruktor/29-1-0-946.

11 Сервер автоматизации [Электронный ресурс] : В помощь Веб-Мастеру / под ред. А.Д. Беляева. – М.: Alexander D. Belyaev, 2005-2014. – Режим доступа :

http://wm-help.net/books-online/book/56472/56472-89.html.