Министерство образования и науки Российской Федерации

Университетский колледж
федерального государственного бюджетного образовательного учреждения
высшего образования
«Оренбургский государственный университет»

Предметно-цикловая комиссия информационных технологий

Н.А. Уйманова, М.Г. Таспаева

ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Практикум

Рекомендовано ученым советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» для обучающихся по программам среднего профессионального образования по специальности 09.02.03 Программирование в компьютерных системах

Оренбург
2017

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

Рецензент – кандидат технических наук, доцент А.А. Попов

Уйманова, Н.А.

У 35 Основы объектно-ориентированного программирования [Электронный ресурс]: практикум / Н.А.Уйманова, М.Г.Таспаева; Оренбург. гос. ун-т. - Электрон. дан. - Оренбург : ОГУ, 2017. - 1 электрон. опт. диск (CD-ROM) : зв., цв. ; 12 см. - Систем. требования : IBM PC 686 (Pentium или выше) ; Microsoft Windows XP ; 512 Мб ; монитор, поддерживающий режим 1024x768 ; мышь или аналогич. устройство - Загл. с этикетки диска.

ISBN 978-5-7410-1993-1

Практикум содержит теоретическую справку, варианты заданий для выполнения лабораторных работ, обеспечивающих учебный процесс по дисциплине «Основы объектно-ориентированного программирования».

Практикум предназначен для обучающихся по программам среднего профессионального образования по специальности 09.02.03 Программирование в компьютерных системах.

Содержание

Введение

Учебная дисциплина «Основы объектно-ориентированного программирования» является дисциплиной из вариативной части программы подготовки специалистов среднего звена, 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 также 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.

Практикум предназначен для проведения лабораторных занятий по дисциплине «Основы объектно-ориентированного программирования» по темам «Основные понятия объектно-ориентированного программирования», «Знакомство со средой Delphi» и «Прямая работа с файлами в среде Delphi», «Понятие виртуального метода»,

«Динамическое распределение памяти», «Работа с диаграммами и таблицами в Delphi», «Исключительные ситуации», «Сервер автоматизации Microsoft Word» и «Работа с базами данных в Delphi». Рассмотренные лабораторные работы позволят студентам понять практические основы проектирования, используя объектно-ориентированный синтаксис программного средства. А также изучить основные принципы работы в среде программирования Delphi, что послужит базой для написания более сложных с точки зрения программирования и проектирования интерфейса программных продуктов.

1 Лабораторная работа №1. Создание простейшего класса объектов. Простое наследование. Родительские и дочерние классы

1.1 Цель работы

Научиться описывать классы объектов, организовывать механизм простого наследования.

1.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Создать проект в среде визуального программирования Delphi (запустив приложение Delphi или выполнить пункт меню в открытом окне среды File->New->Application);

3 В секции interface программного модуля unit unit1 описать класс Tfigura ниже описания типа Tform1 (см.методические рекомендации к лабораторной работе);

4 Аналогичным образом описать дочерний класс Tline;

5 В разделе объявления переменных var, описать объект типа Tline (см.методические рекомендации);

6 В секции implementation написать реализацию методов Draw, описанного в классе Tfigura и метода Resize, описанного в классе Tline (см. методические рекомендации);

7 Поместить на форму объект Button1;

8 Изменить свойство Caption данного объекта на «Нарисовать фигуру»;

9 Организовать для объекта Button1 событие OnClick;

10 Внутри образовавшегося метода осуществить вызов методов Draw и Resize (см. методические рекомендации);

11 В разделе var объявить переменные xn,yn типа integer и переменную cl типа Tcolor;

12 Сохранить проект в директории на диске с именем Lab1_1 (File->Save Project as…), тип фалов при сохранении не менять;

13 Провести отладку и компиляцию проекта (клавиша F9);

14 Создать проект в среде визуального программирования Delphi, реализовать описание класса согласно индивидуального задания (создать единственный класс объектов с именем фигуры, например для объекта окружность класс Tcircle. Класс содержит один из двух методов, указанных в индивидуальном задании);

15 Организовать реализацию данного метода в секции implementation;

16 Сохранить проект на диске в директории Lab1_2;

17 Провести отладку и компиляцию проекта;

18 Оформить отчет о проделанной работе;

Задание повышенного уровня:

19 Для своего индивидуального задания разработать родительский класс Tfigura;

20 Реализовать механизм простого наследования, где родительский класс имеет имя Tfigura, дочерний класс – класс описывающий объект индивидуального задания;

21 Выполнить порождение 10 объектов описанного дочернего класса (по щелчку на кнопку осуществить прорисовку объекта с измененными свойствами, согласно методов индивидуального задания. Например, методы изменение цвета контура и положения объекта по горизонтали – объект каждый раз меняет свое положение по оси X, изменяя при этом цвет контура);

22 Сохранить проект;

23 Выполнить отладку и компиляцию проекта;

24 Оформить отчет о проделанной работе;

25 Защитить работу.

1.3 Содержание отчета

1 Цель, ход работы;

2 Постановка задачи №1, 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 Что называется объектно-ориентированным программированием? В чем основное отличие объектно-ориентированной программы от модульной или линейной?

2 В чем состоит отличие типа «объект» от типа «запись»?

3 Что называется объектом, свойством, методом, классом? Приведите примеры программного обращения к указанным понятиям.

4 Что такое инкапсуляция? Приведите пример описания объектного типа.

1.5 Методические рекомендации

1.5.1 Объектный тип данных. Описание класса

Тип объект (*Object*) можно рассматривать как усовершенствование типа запись (*Record*): описание свойств, параметры моделируемой сущности (они занимают поля объекта) дополняются методами – описаниями действий с объектом. Собственно, в описании объектного типа дают лишь заголовки соответствующих блоков, а сами блоки приводят ниже. Имеется и еще одно чисто формальное

отличие от описания записи: вместо *Record* используется слово *Object*. Переменные объектного типа называются экземплярами объекта. В отличие от типа «запись» объектный тип содержит не только поля, описывающие данные, но также процедуры и функции, описания которых содержаться в описании объекта. Эти процедуры и функции и являются методами. Методам объекта доступны его поля. Методы и их параметры определяются в описании объекта, а их реализация дается вне этого описания, в том месте программы, которое предшествует вызову данного метода. В описании объекта содержатся лишь шаблоны обращений к методам, которые необходимы компилятору для проверки соответствия количества параметров и их типов при обращении к методам.

Описание объектного типа осуществляется в секции *interface* программного модуля (рисунок 1).

unit Unit1;

interface

…

type

TFigura=class ⟶ класс объекта

SFigur:real; → свойство объекта

Рисунок 1 – Описание объектного типа Tfigura

Реализация описанных в объектном типе методов происходит в секции *implementation* программного модуля. Имя метода является составным: вначале указывается тип объекта, а затем сам метод.

….

Implementation

Procedure ***Tfigura.Draw****(c:Tcolor);*

Begin

// тело метода;

End;

1.5.2 Графические возможности Delphi. Свойство Canvas

Работа с графикой в Delphi предполагает обращение к свойству *Canvas* компонентов. Для программиста *Canvas* в Delphi – это холст (канва), который дает доступ к каждому пикселю. Конечно, рисовать попиксельно для работы с графикой в Delphi не приходится, система Delphi предоставляет мощные средства работы с графикой, облегчающие задачу программиста.

В работе с графикой в Delphi в распоряжении находится канва (свойство *Canvas* Delphi компонентов), кисть(свойство *Brush*), карандаш(свойство *Pen*) компонента, на котором планируется рисовать. У *Pen* и *Brush* есть свойство *Color*(цвет) и *Style*(стиль). Чтоб получить доступ к шрифтам, предоставлено свойство холста(*Canvas*) *Font*.

Таблица 1 – Свойства объекта *Pen* (Карандаш)

Свойство	Описание
Color	Цвет линии
Width	Толщина линии (задается в пикселах)
Style	Вид линии: psSolid — сплошная; psDash — пунктирная, длинные штрихи; psDot – пунктирная, короткие штрихи; psDashDot — пунктирная, с чередованием длинного и короткого штрихов; psDashDotDot — пунктирная с чередованием одного длинного и двух коротких штрихов; psClear — линия не отображается

Таблица 2 – Свойства объекта *Brush*(Кисть)

Свойство	Описание
Color	Цвет закраски замкнутой области.
Style	Стиль закраски области:— сплошная заливка; штриховка: bsHorizontal—горизонтальная; bsVertical—вертикальная; bsFDiagonal — диагональная с наклоном линии вперед; bsBDiagonal — диагональная с наклоном линии назад; bsCross—диагональная клетка.

Таблица 3 – Основные свойства класса *Tfont*

Свойство	Описание
Name	Шрифт, который используется для отображения текста. В качестве значения следует брать название шрифта, например Arial.
Size	Размер шрифта
Style	Стиль начертания символов. Задается с помощью констант: fsBold(полужирный), fsItalic(курсив), fsUnderline(подчеркнутый), fsStrikeOut(перечеркнутый). Свойство Style является множеством, что позволяет комбинировать необходимые стили. Например, инструкция, которая устанавливает стиль «полужирный курсив», выглядит так: Font.Style := [fsBold, fsItalic]
Color	Цвет символов. В качестве значения можно использовать константу типа Tcolor.

Не все компоненты в Delphi имеют эти свойства. Свойство *Canvas* имеют,

например, такие компоненты как *ListBox, ComboBox, StringGrid*, а также и сама форма, которая размещает компоненты.

Холст для рисования *Canvas*, представляет собой перевернутую систему координат XOY, где O эта верхняя левая точка компонента или формы.

Таблица 4 – Константы Tcolor

Цвет	Константа
Бирюзовый	clAqua
Черный	clBlack
Синий	clBlue
Ярко-розовый	clFuchsia
Зеленый	clGreen
Салатовый	cLime
Каштановый	clMaroon
Темно-синий	clNavy
Оливковый	clOlive
Фиолетовый	clPurple
Красный	clRed
Серебристый	clSilver
Зелено-голубой	clTeal
Белый	clWhite

Основное свойство такого объекта как Canvas Delphi – Pixels[i,j] типа *Tcolor*, то есть это двумерный массив точек (пикселов), задаваемых своим цветом.

Рисование на канве происходит в момент присвоения какой-либо точке канвы заданного цвета. Каждому пикселю может быть присвоен любой доступный для Windows цвет.

Form1.Canvas.Pixels[100, 100] := clRed;

приведёт к рисованию красной точки с координатами [100, 100].

Узнать цвет пиксела можно обратным присвоением:

Color :=Form1.Canvas.Pixels[100,100];

Таблица 5 – Свойства класса *Tcanvas*

Свойство	Описание
Pen	Карандаш. Определяет цвет, стиль и толщинц линии, которую рисует, например метод Lineto
PenPos	Положение (координаты) карандаша
Brush	Кисть. Определяет цвет и стиль закраски области, например прямоугольника, который рисует метод Rectangle.
Font	Шрифт. Определяет шрифт, который используется для вывода текста, например методом TextOut.

Таблица 6 – Методы класса *Tcanvas*

Метод	Действие
1	2
Lineto(x, y)	Рисует линию из текущей точки в точку с указанными координатами (перемещение указателя текущей точки в нужную обеспечивает метод Moveto). Цвет линии определяется свойством Pen.Color .
Rectangle(x1 ,y1 ,x2, y2)	Рисует прямоугольник. Параметры x1, y1указывают координаты верхней левой точки, ax2, y2 координаты нижней правой точки. Цвет границы прямоугольника определяет свойство Pen.Color , цвет закраски области — свойствоBrush.Color .

Продолжение таблицы 6

1	2
RoundRect(x1, y1, x2, y2, x3, y3)	Рисует прямоугольник со скругленными углами. Параметры x1, y1 указывают координаты верхней левой точки, x2, y2 координаты нижней правой точки, а x3, y3 радиус скругления. Цвет границы прямоугольника определяет свойство Pen.Color ,цвет закраски области — свойство Brush.Color .
Ellipse(x1, y1, x2, y2)	Рисует эллипс (окружность). Параметры x1, y1 указывают координаты верхней левой точки, ax2, y2 координаты нижней правой точки прямоугольника в который вписана окружность. Цвет границы прямоугольника определяет свойство Pen.Color , цвет закраски области — свойство Brush.Color .
Arc(x1, y1, x2, y2, x3, y3,x4, y4)	Рисует дугу. Параметры x1, y1, x2 и y2 задают эллипс, частью которого является дуга, параметры x3, y3, x4 и y4 — начальную и конечную точку дуги. Цвет дуги определяет свойство Pen.Color.
Pie(x1 ,y1 ,x2 ,y2 ,x3 ,y3 ,x4 ,y4)	Рисует сектор. Параметры x1, y1, x2 и y2 задают эллипс, частью которого является сектор, параметры x3, y3, x4 и y4 —границы сектора. Цвет границы сектора определяет свойствоPen.Color , цвет закраски сектора — свойствоBrush.Color .
FillRect(aRect)	Рисует закрашенный прямоугольник. Параметр aRect(тип Trect) определяет положение и размер прямоугольника. Цвет закраски области определяет свойство Brush.Color .

Продолжение таблицы 6

1	2
FrameRect(aRect)	Рисует контур прямоугольника. Параметра Rect (тип Trect) определяет положение и размер прямоугольника. Цвет контура определяет свойство Brush.Color .
Polyline(points, n)	Рисует ломаную линию. Points — массив типаTPoint. Каждый элемент массива представляет собой запись, поля x и y, которые содержат координаты точки перегиба ломаной. N —количество звеньев ломаной. Метод Polyline вычерчивает ломаную линию, последовательно соединяя прямые точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т.д.

Вычерчивание графических элементов на графической поверхности, например компонента *Image* (закладка *Additional*), выполняют соответствующие методы класса *Tcanvas*.

Инструкция, обеспечивающая вычерчивание графического элемента, в общем виде выглядит так:

Объект.Canvas.*Метод (Параметры)*;

Объект определяет объект, на поверхности которого нужно нарисовать графический элемент. В качестве объекта можно указать компонент *Image.*

Метод – это имя метода, который обеспечивает рисование нужного графического элемента.

Параметры, в большинстве случаев, определяют положение графического элемента на графической поверхности и его размер.

Например, в результате выполнения инструкции

Form1.Canvas.Rectangle(10,20,60,40);

В поле компонента *Form1* будет нарисован прямоугольник шириной 50 и высотой 20 пикселей, левый верхний угол которого будет находиться в точке(10,20).

При записи инструкций, обеспечивающих вывод графики, удобно использовать инструкцию *with,* которая позволяет сократить количество набираемого кода.

Вывод строки текста на графическую поверхность объекта обеспечивает метод *TextOut*. Инструкция вызова метода *TextOut* в общем виде выглядит следующим образом:

Объект.Canvas.TextOut(x, y, '*Текст*');

Параметры *x, y* определяют координаты точки графической поверхности, от которой выполняется вывод текста.

Шрифт, используемый для отображения текста, определяет свойство Font графической поверхности, на которую текст выводится. Свойство *Font* представляет собой объект типа Tfont.

1.5.3 Создание простого класса фигур

Для начала работы над проектом необходимо запустить среду программирования Delphi.

Внутри программного модуля описать объектный тип данных Tfigura

```
type
Tfigura=class
procedure Draw (c:Tcolor);
end;
```

Организовать дочерний тип данных

```
Tline=class(Tfigura)
procedure Resize(xx,yy:integer);
end;
```

В глобальном разделе переменных объявить объект класса Tline

var

Form1: Tform1;

L:Tline;

В секции implementation организовать реализацию описанных методов

implementation

*{$R *.dfm}*

procedureTFigura.Draw(c:Tcolor);

begin

form1.canvas.pen.color:=c;

end;

procedureTLine.Resize(xx,yy:integer);

begin

form1.canvas.LineTo(xx,yy);

end;

Окно приложения Delphi состоит из нескольких окон. В окно формы необходимо поместить со вкладки *Standart* объект *Button1* (рисунок 2).

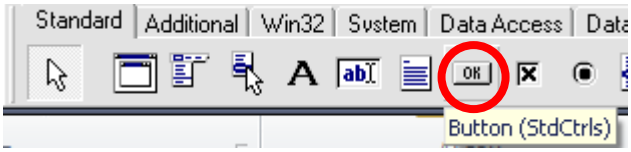


Рисунок 2 – Вкладка *Standart*

Для объекта *Button1*изменить свойство *Caption* на «*Нарисовать объект*»

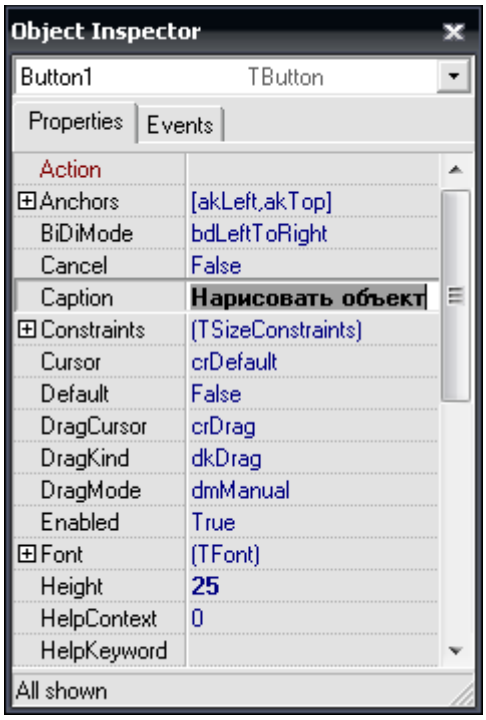


Рисунок 3 – Свойства объекта Button1

Для объекта Button1 организовать событие OnClick, прописать следующий код:

```
procedure Tform1.Button1Click(Sender: Tobject);
begin
xn:=40;
yn:=50;
cl:=clRed;
L.Draw(cl);
L.Resize(xn,yn);
end;
```

1.6 Варианты индивидуальных заданий

1 Класс объектов «Окружность». Методы изменение цвета контура и радиуса;

2 Класс объектов «Квадрат». Методы рисование квадрата, изменение его положения (по горизонтали);

3 Класс объектов «Прямоугольник». Методы изменение цвета контура и размера;

4 Класс объектов «Скругленный прямоугольник». Методы изменение размера и контура фигуры;

5 Класс объектов «Окружность». Методы изменение цвета заливки и положения (по вертикали);

6 Класс объектов «Прямоугольник». Методы изменение стиля заливки и положения (по горизонтали);

7 Класс объектов «Квадрат». Методы изменение стиля заливки и цвета контура;

8 Класс объектов «Сектор». Методы Изменение цвета контура и стиля заливки;

9 Класс объектов «Скругленный прямоугольник». Методы изменение размеров и цвета контура;

10 Класс объектов «Сектор». Методы изменение размеров и цвета заливки;

11 Класс объектов «Окружность». Методы изменение положения и стиля контура;

12 Класс объектов «Овал». Методы изменение горизонтального радиуса и цвета заливки;

13 Класс объектов «Квадрат». Методы изменение размера и цвета заливки;

14 Класс объектов «Овал». Методы изменение положения (по горизонтали) и изменение стиля контура;

15 Класс объектов «Прямоугольник». Методы изменение положения (по вертикали) и цвета заливки.

2 Лабораторная работа №2. «Delphi – среда визуального программирования»

2.1 Цель работы

Познакомиться с основными элементами визуальной среды программирования Delphi, овладеть приемами работы с формами.

2.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Задание 1

– создать проект в среде визуального программирования Delphi, демонстрирующий простейшие возможности среды (внешний интерфейс формы проекта представлен на рисунке 5);

– сохранить проект в директории на диске с именем Lab2_1;

– провести отладку и компиляцию проекта;

– оформить отчет о проделанной работе;

3 Задание 2

– создать проект в среде визуального программирования Delphi, демонстрирующий операцию сложения двух чисел (внешний интерфейс формы проекта представлен на рисунке 7);

– сохранить проект на диске в директории Lab2_2;

– провести отладку и компиляцию проекта;

4 Задание 3

– создать проект в среде визуального программирования Delphi, демонстрирующий программный доступ к свойствам формы, (внешний интерфейс формы проекта представлен на рисунке 8);

– сохранить проект в директории на диске с именем Lab2_3;

– провести отладку и компиляцию проекта;

– оформить отчет о проделанной работе;

5 Задание 4

– создать проект в среде визуального программирования Delphi, демонстрирующий реакцию на события формы (на форме разместить компонент Label);

– сохранить проект в директории на диске с именем Lab2_4;

– провести отладку и компиляцию проекта;

– оформить отчет о проделанной работе;

6 Задание 5

– создать заставку к существующему проекту согласно методическим рекомендациям п. 2.5.4;

– провести отладку и компиляцию проекта;

7 Задание 6

– создать иконку к существующему проекту согласно методическим рекомендациям, указанных в п.2.5.5;

– провести отладку и компиляцию проекта;

8 Задание 7. Добавить в репозиторий Delphi ранее созданную форму в соответствии с методическими рекомендациями п. 2.5.5;

9 Оформить отчет о проделанной работе;

Задание повышенного уровня:

1 Создать проект в среде визуального программирования Delphi, демонстрирующий программный доступ к свойствам формы, реализовать реакцию на события формы согласно своего индивидуального задания;

2 Сохранить проект;

3 Провести отладку и компиляцию проекта;

4 Оформить отчет о проделанной работе;

5 Защитить работу.

2.3 Содержание отчета

1 Цель, ход работы;

2 Постановка задачи заданий 1-7, листинг программного модуля, результат работы приложения;

3 Формулировка индивидуального задания;

4 Листинг программного модуля индивидуального задания, результат работы приложения;

5 Вывод.

2.4 Контрольные вопросы

1 Какие элементы включает интерфейс среды разработки Delphi?

2 Какие файлы входят в состав проекта Delphi?

3 Перечислите основные свойства формы. Как программно изменить свойства?

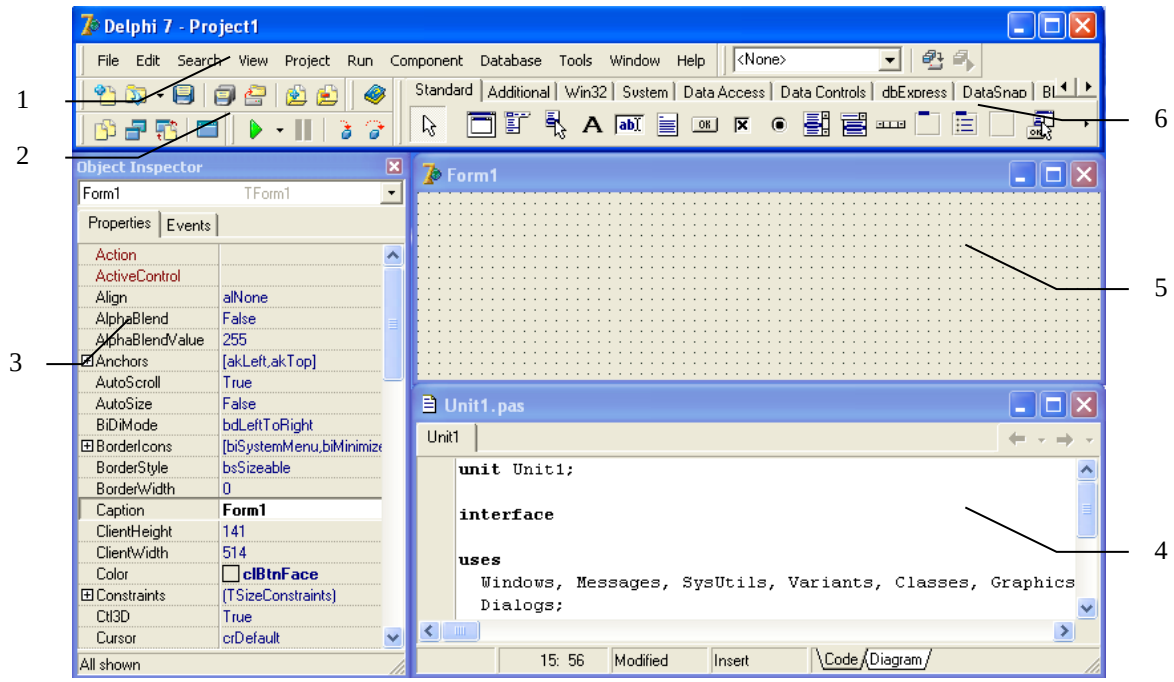
4 Как настроить опции проекта?

5 Что такое репозиторий Delphi? Как добавить форму в репозиторий?

2.5 Методические рекомендации

2.5.1 Создание простейшего проекта

Основные элементы интерфейса интегрированной среды разработки Delphi представлены на рисунке 4.



1 – основное меню; 2 – панель инструментов;

3 – окно инспектора объектов; 4 – редактор кода программы;

5- окно формы; 6 – палитра компонентов.

Рисунок 4 – Основные элементы интерфейса

Основное меню содержит следующие команды: File, Edit, Search, View, Project, Run, Component, Database, Tools, Window, Help.

Палитра компонентов содержит множество компонентов, которые подразделяются на несколько групп. Каждая группа размещена на своей странице палитры компонентов.

Окно инспектора объектов предназначено для изменения свойств выбранных компонентов и состоит из двух страниц. Страница Properties (Свойства) предназначена для изменения необходимых свойств компонента, страница Events (События) – для определения реакции компонента или формы на то или иное событие (например, щелчок «мыши» на кнопке – событие OnClick, создание формы – OnCreate).

Окно формы представляет собой проект Windows-окна программы. На этом окне в процессе написания программы размещаются необходимые компоненты.

Редактор кода программы предназначен для просмотра, написания и редактирования текста программы. В системе Delphi используется язык программирования Object Pascal.

Программа в среде DELPHI составляется как описание алгоритмов, которые будут выполняться, если возникает определенное событие, связанное с формой или с каким-либо из размещенных на ней компонентов. Для каждого обрабатываемого события, с помощью страницы Events инспектора объектов в тексте программы организуется процедура (procedure), между ключевыми словами begin и end которой программист записывает на языке Object Pascal требуемый алгоритм.

Переключение между окном формы и окном редактора кода осуществляется с помощью клавиши F12.

Задание 1.

1 Создаем новый проект приложения, выбрав команду File-New, а затем Application;

2 Выберем команду *File-Save Project As* и в появившемся окне выберем новую папку, введем имя сохраняемого модуля MyUnit, а в следующем окне – имя сохраняемого проекта, т.е. будущей программы Program1;

3 Создадим простейшее приложение:

Разместим на форме 2 компонента *Label*  вкладки *Standart*. В окне инспектора объектов у объекта *Label1* изменим значение свойства *Caption* с «*Label1*» на *Меня зовут*. У объекта *Label2* удаляем значение свойства *Caption*.

Разместим на форме компонент *Edit*  вкладки *Standart*. В окне инспектора объектов у объекта *Edit1* свойстве Text удалите значение «*Edit1*» (т.е. задаем свойству *Text* значение, равное пустой строке).

Разместим на форме кнопку *Button*  вкладки *Standart*. Изменим свойство *Caption* компонентов *Button1* на *Привет!*

Форма должна принять следующий вид (рисунок 5):

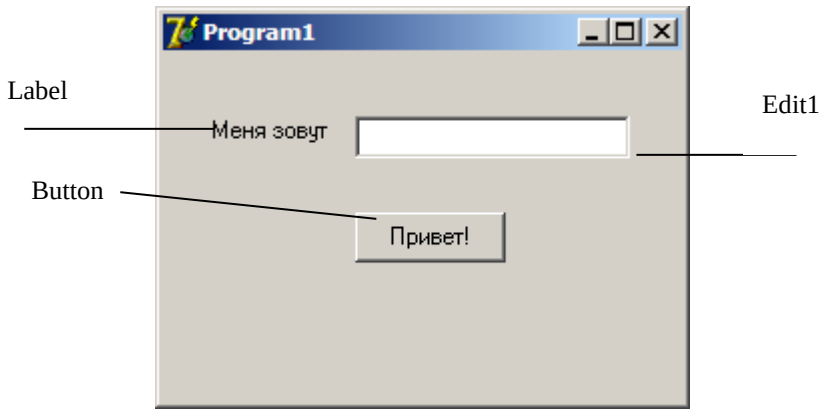


Рисунок 5 – Внешний интерфейс формы

Создадим обработчик события нажатия кнопки. При нажатии на кнопку «Привет!» программа должна вывести в метку *Label2* текст. Для создания обработчика события нажатия этой кнопки нужно выделить компонент Button1, в инспекторе объектов перейти на закладку Events, и дважды щелкнуть в строке напротив названия события **OnClick**. В модуле программы появится заготовка процедуры обработчика события:

```
procedure Tform1.Button1Click(Sender: Tobject);
begin
end;
```

А в описании класса формы добавится поле с заголовком процедуры:

```
procedure Button1Click(Sender: Tobject);
```

Приведем эту процедуру к следующему виду:

```
procedure Tform1.Button1Click(Sender: Tobject);
begin
  Label2.Caption:='Здравствуй, '+Edit1.Text+'!';
end;
```

4 Запустить программу можно выбрав команду *Run* в меню *Run*, или нажав клавишу F9. При этом происходит компиляция и, если нет ошибок, компоновка

программы и создание единого загружаемого файла с расширением .exe. На экране появляется главная форма программы;

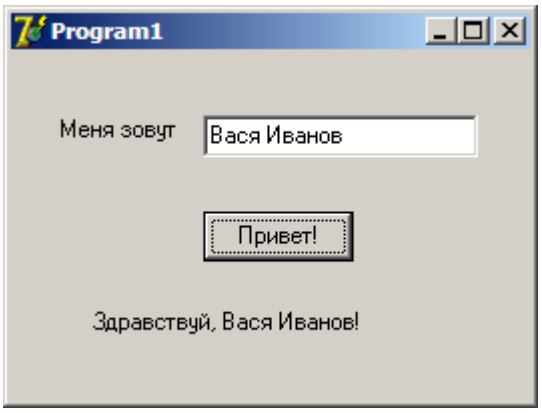


Рисунок 6 – Результат выполнения программы

Задание 2.

1 Создаем новый проект приложения, выбрав команду File-New, а затем Application;

2 Выберем команду *File-Save Project As* и в появившемся окне выберем новую папку, введем имя сохраняемого модуля MyUnit, а в следующем окне – имя сохраняемого проекта, т.е. будущей программы MyProgram;

3 Создадим приложение, которое будет суммировать два числа и выводить ответ в метку Label:

Разместим на форме 2 компонента *Label*  вкладки *Standart*. В окне инспектора объектов у объектов *Label1* и *Label2* изменим значение свойства Caption с «*Label1*» и «*Label2*» на *Число a* и *Число b* соответственно.

Разместим на форме 2 компонента *Edit*  вкладки *Standart*. В окне инспектора объектов у объектов *Edit1* и *Edit2* в свойстве Text удалите значения «*Edit1*» и «*Edit2*» (т.е. задаем свойству *Text* значение, равное пустой строке).

На форме также необходимо разместить кнопку *Button*  вкладки *Standart*. Изменим свойство *Caption* компонентов *Button1* на *Вычислить*.

Для вывода результата сложения двух чисел добавим на форму еще одну метку *Label* (значение свойства Caption – *Сумма равна*)

Форма должна принять следующий вид:

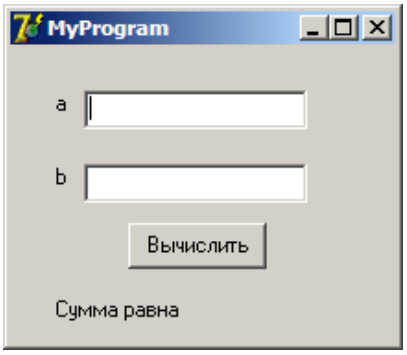


Рисунок 7 – Внешний интерфейс формы проекта

Создадим обработчик события нажатия кнопки. При нажатии на кнопку «Вычислить» программа должна производить необходимые вычисления и выводить результат. Для создания обработчика события нажатия этой кнопки нужно выделить компонент Button1, в инспекторе объектов перейти на закладку Events, и дважды щелкнуть в строке напротив названия события *OnClick*. В модуле программы появится заготовка процедуры обработчика события:

```
procedure Tform1.Button1Click(Sender: Tobject);
begin
end;
```

А в описании класса формы добавится поле с заголовком процедуры:

```
procedure Button1Click(Sender: Tobject);
```

Приведем эту процедуру к следующему виду:

```
procedure Tform1.Button1Click(Sender: Tobject);
  var a,b,c:integer;
begin
  a:=StrToInt(Edit1.Text);
  b:= StrToInt(Edit2.Text);
```

c:=a+b;

Label3.Caption:=Label3.Caption+IntToStr©;

end;

Здесь после слова *var* перечисляются имена переменных величин и указывается их тип. Мы используем переменные *a, b* и *c*, которые имеют целый тип.

Функция *StrToInt()* выполняет преобразование строки (в данном случае хранящейся в поле *Edit*) в целое число (такое преобразование возможно, если строка состоит только из цифр).

Функция *IntToStr()* выполняет обратное преобразование, т.е. преобразует целое число в строку.

Запустить программу можно выбрав команду *Run* в меню *Run*, или нажав клавишу F9. При этом происходит компиляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением .exe. На экране появляется главная форма программы.

2.5.2. Работа со свойствами формы. Программный доступ к свойствам формы

Форма - это компонент Form класса Tform. На ее основе конструируется приложение. Существуют 2 класса форм:

1 Немодальные – те, которые позволяют переключаться в другую форму приложения без своего закрытия;

2 Модальные – те, которые требуют обязательного закрытия перед обращением к другой форме.

Каждое приложение имеет одну главную форму и, возможно, несколько второстепенных. Главная форма загружается автоматически при запуске приложения.

Каждая форма имеет две области:

1 Клиентская область – та часть формы, в которой размещаются визуальные компоненты;

2 Неклиентская область – занята рамкой, заголовком формы и строкой главного меню.

Ниже перечислены Delphi свойства формы (объекта Tform):

- *Name* – Имя формы. Испльзуется для управления формой и доступа к копонентам формы;
- *Caption* – Текст заголовка;
- *Widht* – Ширина Delphi свойств формы;
- *Height* – Высота формы;
- *Top* – Расстояние от верней части формы до верхней границы экрана;
- *Left* – Расстояние от левой границы формы до левой границы экрана;
- *Visible* – позволяет скрывать и отображать данную форму;
- *Active* – определяет активность формы;
- *Clientwidth* – возвращает ширину клиентской области;
- *Clientheight* – возвращает высоту клиентской области;
- *Windowstate* – определяет состояние отображаемой формы;
- *BorderStyle* – Вид границы окна формы. Она может быть обычной (bsSizeable), тонкой (bsSingle) или отсутствовать (bsNone);
- *BorderIcons* – Кнопки управления окном. Значение их свойств определяет, какие кнопки управления окном будут доступны пользователю во время работы программы;
- *Icon* – Значок в заголовке окна, обозначающего кнопку вывода системного меню;
- *Color* – Цвет фона. Его можно задать, указав название цвета или сделать привязку к цветовой схеме операционной системы;
- *Font* – Шрифт, то есть его можно выбрать из диалогового окна;
- *AlphaBlend* – использовать ли прозрачность формы;
- *AlphaBlendValue* — степень прозрачности формы (0-прозрачна полностью, 255-непрозрачна).

Задание 3

Создать приложение, программно изменяющее свойства формы и имеющее интерфейс, представленный на рисунке 8.

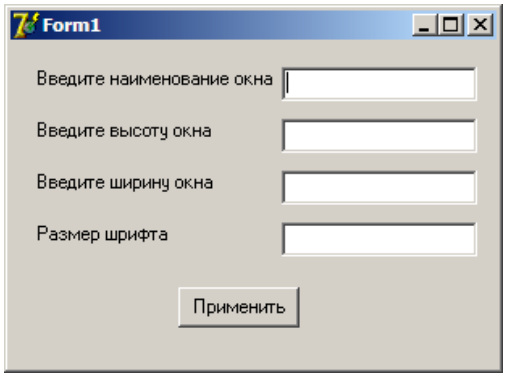


Рисунок 8 – Внешний интерфейс формы проекта

1 Создаем новый проект приложения, выбрав команду File-New, а затем Application;

2 Выберем команду *File-Save Project As*;

3 Размещаем на форме необходимые компоненты;

4 Создадим обработчик события нажатия кнопки. Приведем процедуру обработчика событий к следующему виду:

```
procedure Tform1.Button1Click(Sender: Tobject);
begin
  form1.Caption:=edit1.Text;
  form1.Height:=StrToInt(edit2.Text);
  form1.Width:=StrToInt(edit3.Text);
  form1.Font.Size:=StrToInt(edit4.Text);
end;
```

5 Запустить программу можно выбрав команду *Run* в меню *Run*, или нажав клавишу F9. При этом происходит компиляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением .exe. На экране появляется главная форма программы.

2.5.3 Организации реакции на события формы

События формы

- OnCreate — происходит сразу после создания формы. Обработчик этого события может установить начальные значения для свойств формы и ее компонентов, запросить у операционной системы необходимые ресурсы, создать служебные объекты, а также выполнить другие действия прежде, чем пользователь начнет работу с формой. Парным для события OnCreate является событие OnDestroy;
- OnDestroy — происходит непосредственно перед уничтожением формы. Обработчик этого события может освободить ресурсы, разрушить служебные объекты, а также выполнить другие действия прежде, чем объект формы будет разрушен;
- OnShow — происходит непосредственно перед отображением формы на экране. Парным для события OnShow является событие OnHide;
- OnHide — происходит непосредственно перед исчезновением формы с экрана. Парным для события OnHide является событие OnShow;
- OnActivate — происходит, когда пользователь переключается на форму, т.е. форма становится активной. Парным для события OnActivate является событие OnDeactivate;
- OnDeactivate — происходит, когда пользователь переключается на другую форму, т.е. текущая форма становится неактивной. Парным для события OnDeactivate является OnActivate;
- OnCloseQuery — происходит при попытке закрыть форму. Запрос на закрытие формы может исходить от пользователя, который нажал на рамке формы кнопку «Закрыть», или от программы, которая вызвала у формы метод Close. Обработчику события OnCloseQuery передается по ссылке булевский параметр CanClose, разрешающий или запрещающий действительное закрытие формы;
- OnClose — происходит после события OnCloseQuery, непосредственно перед закрытием формы;

- OnContextPopup — происходит при вызове контекстного меню формы;
- OnMouseDown — происходит при нажатии пользователем кнопки мыши, когда указатель мыши наведен на форму. После отпускания кнопки мыши в компоненте происходит событие OnMouseUp. При перемещении указателя мыши над формой периодически возникает событие OnMouseMove, что позволяет отслеживать позицию указателя;
- OnMouseWheelUp — происходит, когда колесико мыши проворачивается вперед (от себя);
- OnMouseWheelDown — происходит, когда колесико мыши проворачивается назад (на себя);
- OnMouseWheel — происходит, когда колесико мыши проворачивается в любую из сторон;
- OnStartDock — происходит, когда пользователь начинает буксировать стыкуемый компонент;
- OnGetSiteInfo — происходит, когда стыкуемый компонент запрашивает место для стыковки;
- OnDockOver — периодически происходит при буксировке стыкуемого компонента над формой;
- OnDockDrop — происходит при стыковке компонента;
- OnEndDock — происходит по окончании стыковки компонента;
- OnUnDock — происходит, когда пользователь пытается отстыковать компонент.
- OnDragDrop — происходит, когда пользователь опускает в форму буксируемый объект;
- OnDragOver — периодически происходит при буксировке объекта над формой;
- OnCanResize — происходит при попытке изменить размеры формы. Запрос на изменение размеров может исходить от пользователя. Обработчику события OnCanResize передается по ссылке булевский параметр Resize, разрешающий или запрещающий действительное изменение размеров формы;

- OnResize — происходит при изменении размеров формы;
- OnConstrainedResize — происходит при изменении размеров формы и позволяет на лету изменять минимальные и максимальные размеры формы;
- OnShortCut — происходит, когда пользователь нажимает клавишу на клавиатуре. Позволяет перехватывать нажатия клавиш еще до того, как они дойдут до стандартного обработчика формы;
- OnKeyDown – генерируется, когда нажата клавиша на клавиатуре;
- OnKeyPress – генерируется, когда нажата и отпущена клавиша на клавиатуре;
- OnKeyUp – генерируется, когда отпущена клавиша на клавиатуре.

Задание 4

Создать приложение, демонстрирующее реакцию на события формы.

1 Создаем новый проект приложения, выбрав команду File-New, а затем Application;

2 Выберем команду *File-Save Project As*;

3 Создадим обработчик события нажатия клавиши клавиатуры. Приведем процедуру обработчика событий к следующему виду:

```
procedure Tform1.FormKeyPress(Sender: Tobject; var Key: Char);
begin
  Label1.Caption:='Вы нажали клавишу!';
end;
```

4 Создадим обработчик события, происходящее при нажатии и отпускании кнопки мыши. Например, нарисуем прямоугольник координаты левого верхнего угла которого будут соответствовать точке нажатия мышки, а координатам левого верхнего угла точка, в которой пользователь отпустил кнопку. Приведем процедуру обработчика событий к следующему виду:

```
var
 Form1: Tform1;
 xx,yy:integer;
...
```

```
procedure Tform1.FormMouseDown(Sender: Tobject; Button: TmouseButton;
  Shift: TshiftState; X, Y: Integer);
begin
   xx:=x;
  yy:=y;
end;

procedure Tform1.FormMouseUp(Sender: Tobject; Button: TmouseButton;
  Shift: TshiftState; X, Y: Integer);
begin
   Form1.Canvas.Rectangle(XX,YY,X,Y);
end;
```

5 Запустить программу можно выбрав команду *Run* в меню *Run*, или нажав клавишу F9. При этом происходит компиляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением .exe. На экране появляется главная форма программы (рисунок 9).

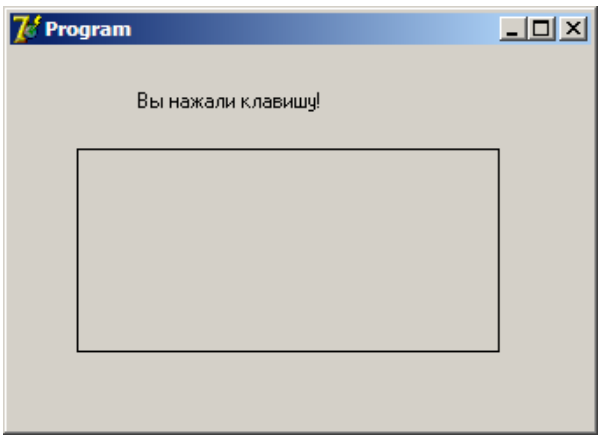


Рисунок 9 – Результат работы приложения

2.5.4 Создание заставки к приложению Delphi

Для создания заставки к приложению Delphi выполняем следующую последовательность действий, описанных в задании 5.

Задание 5.

1 Добавляем в существующий проект форму (*File – New Form*). Это окно и будет заставкой. У него нужно убрать рамку с полосой заголовка, установив свойство *BorderStyle* в *bsNone*;

2 Разрабатываем дизайн окна заставки;

3 Выполняем команду меню *Project- Options*. Во вкладке *Forms* форму *Form2* из списка автоматически создаваемых форм (*Auto-Create forms*) перенести в список доступных форм (*Available forms*);

4 На форме-заставке размещаем компонент *Timer* с вкладки *System*. В его свойстве *Interval* установить значение 5000, а в событии *OnTimer* написать:

Timer1.Enabled := false;

Это сделано для того, чтобы заставка была видна в период указанного времени – 5000 миллисекунд, т.е. 5 секунд;

5 Далее открываем файл проекта: *Project > View Source* и вносим изменения согласно примера ниже:

```
program Project1;
uses
Forms,
Unit1 in 'Unit1.pas' {Form1},
Unit2 in 'Unit2.pas' {IntroForm};
{$R *.res}
begin
Application.Initialize;
  Form2 := Tform2.Create(Application);
  Form2.Show;
  Form2.Update;
```

```
  while Form2.Timer1.Enabled do
    Application.ProcessMessages;
  Application.CreateForm(Tform1, Form1);
  Form2.Hide;
  Form2.Free;
  Application.Run;
end.
```

2.5.5 Настройка опций проекта. Сохранение формы в репозитории Delphi. Отладка проекта

Для установки параметров проекта используется окно параметров проекта (*Project Options*), открываемое командой *Project* → *Options* (Проект → Параметры).

Параметры разбиты на группы, каждая из которых располагается в окне параметров проекта на своей странице.

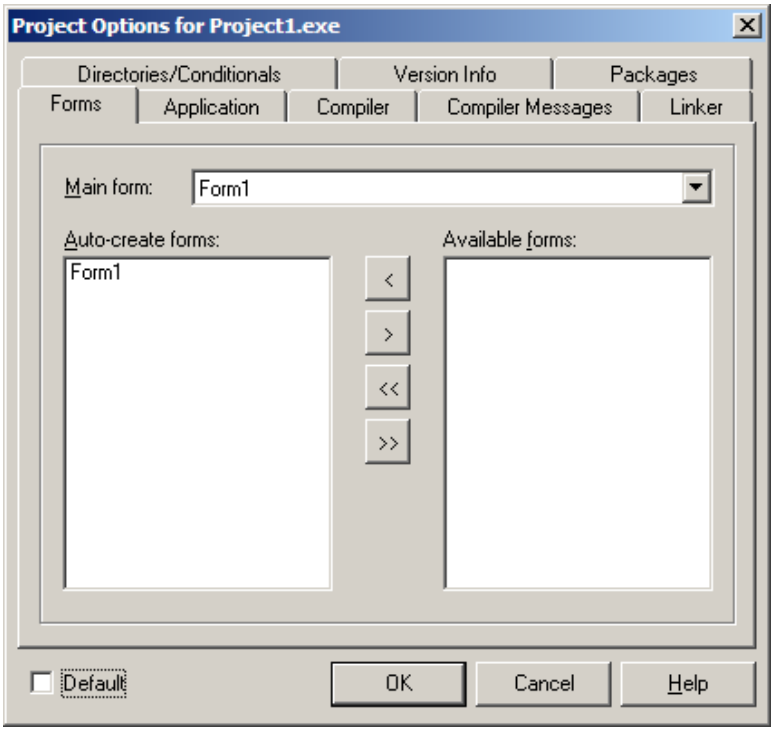


Рисунок 10 – Окно опций проекта

На вкладке *Forms* можно задать главную форму приложения (*Main form*) и в списке *Auto-create forms* указать формы, которые будут создаваться одновременно с главной формой.

На вкладке *Application* можно задать название (*Title*) программы. В среде Delphi дополнительно можно задать файл справки (*Help file*) и значок (*Icon*).

На вкладке *Compiler* настраиваются параметры компилятора. Наиболее интересными из них являются переключатели *Optimization* (включает оптимизацию генерируемого кода) и *Use Debug DCUs* (позволяет отлаживать исходный код системных библиотек). Оба этих переключателя полезны при отладке программы: первый следует выключить, а второй – включить.

На вкладке *Compiler Messages* настраивается чувствительность компилятора к подозрительному коду. Включив переключатели *Show hints* и *Show warnings*, можно получать от компилятора весьма полезные подсказки (*hints*) и предупреждения (*warnings*), а не только сообщения об ошибках.

На вкладке *Linker* настраиваются параметры сборки проекта. Следует обратить внимание на группу *Memory sizes*, особенно на два параметра: *Min stack size* и *Max stack size*. Они задают соответственно минимальный и максимальный размеры стека прикладной программы, т.к. может потребоваться увеличить значения этих параметров при написании приложения, активно использующего рекурсивные подпрограммы.

На вкладке *Directories/Conditionals* можно задать каталоги для различных файлов. Наиболее важные из них: *Output directory* – каталог, в который помещается выполняемый файл; *Unit output directory* – каталог, в который помещаются промежуточные объектные модули (DCU-файлы); *Search path* – список каталогов, в которых осуществляется поиск программных модулей.

На вкладке *Version Info* выводится информация о версии приложения. Для того чтобы эта информация помещалась в выполняемый файл, нужно включить переключатель *Include version information in project*.

На вкладке *Packages* можно управлять списком пакетов, используемых в проекте. Пакеты – это внешние библиотеки компонентов. Переключатель *Build with*

runtime packages позволяет существенно уменьшить размер выполняемого файла за счет использования внешних библиотек компонентов вместо встраивания их кода непосредственно в выполняемый файл. Этот режим выгоден при создании нескольких программ, построенных на базе большого количества общих компонентов.

Задание 6.

Создать иконку для существующего приложения Delphi. Для этого необходимо:

1) запускаем редактор изображений *Image Editor* (меню *Tools- Image Editor*);

2) чтобы начать работу по созданию нового значка, нужно из меню *File* выбрать команду *New*, а из появившегося списка — опцию *Icon File;*

3) после выбора типа создаваемого файла открывается окно *Icon Properties*, в котором необходимо выбрать характеристики создаваемого значка: *size* (размер) — 32x32 (стандартный размер значков Windows) и *Colors* (*Палитра*) — 16 цветов. В результате нажатия кнопки *ОК* открывается окно *Icon1.ico*, в котором можно, используя стандартные инструменты и палитру, нарисовать нужный значок;

4) сохраняем созданный значок (*File-Save*);

5) в среде Delphi выполняем команду *Project → Options*. Во вкладке *Application* по кнопке *Load Icon* загружаем файл созданного значка.

Задание 7.

Окно репозитория Delphi открывается при выборе *File – New – Other* и имеет вид, показанный на рисунке 2.

На его страницах расположены пиктограммы для выбора прототипов форм, модулей, проектов и экспертов построения форм и проектов. Зависимые переключатели Copy, Inherit и Use определяют режим связи между хранящимся в репозитории прототипом и его копией в проекте: *Copy* – выбранный элемент копируется в текущий каталог и автоматически подключается к проекту; между образцом и его копией нет никакой связи; *inherit* – в проекте создаются наследники выбранного элемента и всех его родителей; любые изменения образца проявляются во всех проектах, которые его унаследовали; изменения наследника не влияют на

образец; *Use* – выбранный элемент становится частью проекта; любые его изменения в проекте приводят к изменениям образца и сказываются во всех других проектах, которые его унаследовали или используют.

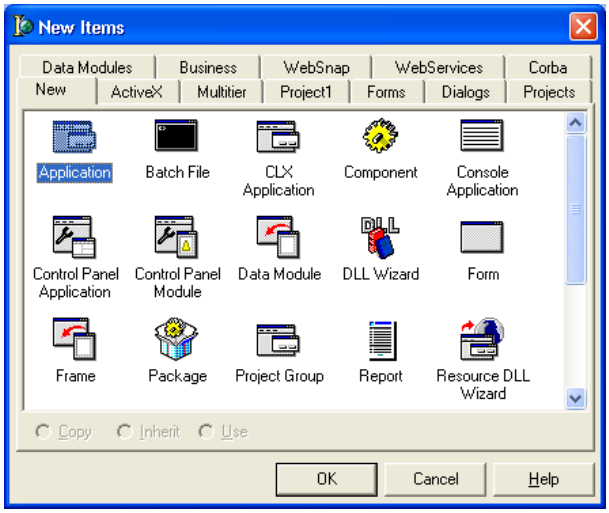


Рисунок 11 – Окно репозитория Delphi

Ниже кратко описывается методика размещения в репозитории разработанной программистом формы.

1 Разработаем форму. Разместим на ней все необходимые интерфейсные компоненты. Сохраним форму;

2 Щелкните по форме правой кнопкой мыши и в локальном меню выберите Add to Repository. Появится диалоговое окно регистрации формы (рисунок 12);

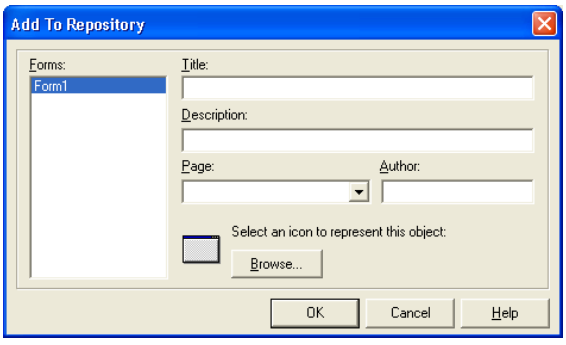


Рисунок 12 – Окно репозитория Delphi

3 В строке *Title* напишем имя, под которым форма будет зарегистрирована в репозитории (*WorkForm)*, в строке *Description* – описание формы, (*Основная форма для проекта MyProgram)*, в строке *Author* – свое имя. Раскроем список *Page* и выберем закладку страницы репозитория *(Forms)*, на которой разместим форму, после чего щелкнем по кнопке *ОК* – форма помещена в репозитории.

2.6 Варианты индивидуальных заданий

1 Создать приложение, на форме которого расположить два текстовых поля, в них задать координаты вывода формы; при двойном нажатии левой кнопки мыши по форме окно выводится в заданном месте;

2 Создать приложение, обработать событие *OnShow,* при котором форма плавно увеличивает ширину до заданного значения. На форме разместить кнопку «Закрыть», при нажатии на которую форма закрывается (воспользоваться циклом for…to…do);

3 Создать приложение, осуществляющее вывод координат указателя мыши в надпись при движении мыши;

4 Создать приложение, при закрытии которого форма становится полупрозрачной и постепенно угасает, закрывая приложение (воспользоваться циклом for…downto…do);

5 Создать приложение. При нажатии правой кнопкой мыши в заголовке формы появляются координаты указателя мыши, при отпускании кнопки мыши заголовок формы очищается;

6 Создать приложение, в котором при изменении размеров формы в заголовок выводятся текущие высота и ширина;

7 Создать приложение. При нажатии клавиши управления курсором на клавиатуре форма смещается в левую, правую сторону, вверх и вниз на заданное расстояние;

8 Создать приложение. При нажатии левой кнопки мыши цвет формы меняется случайным образом. На форме разместить кнопку «Закрыть», при нажатии на которую появляется окно сообщения о подтверждении закрытии формы;

9 Создать приложение. Разместить на форме надпись. При нажатии на клавиатуре стрелки «вверх» размер шрифта надписи увеличивается, при нажатии стрелки «вниз» размер шрифта надписи уменьшается;

10 Создать приложение. Разместить на форме надпись. При нажатии левой кнопки мыши надпись становится невидимой, при отпускании мыши надпись становится видимой;

11 Создать приложение. При двойном щелчке левой кнопки мыши по форме меняется тип рамки формы. Перебрать все возможные варианты обрамления формы;

12 Создать приложение, в котором когда форма активна ее цвет выбирается случайным образом из 256 оттенков красного, при деактивации цвет формы выбирается случайным образом из 256 оттенков зеленого. Чтобы проверить активность формы необходимо в приложении создать еще одну форму (File->New->Form);

13 Создать приложение, в котором при прокручивании колесика мыши вперед размеры формы уменьшались, при прокручивании колесика мыши назад размеры формы увеличивались;

14 Создать приложение. При нажатии левой кнопки мыши по форме в ней появляется «отверстие» круглой формы, при отпускании форма восстанавливается;

15 Создать приложение при щелчке левой кнопки мыши форма плавно уменьшает высоту до фиксированного значения, при двойном щелчке плавно увеличивает высоту до фиксированного значения (воспользоваться циклом for…to…do и for…downto…do).

2.7 Дополнительная литература

1 Свойства и характеристики форм в Делфи [Электронный ресурс]. – Режим доступа: http://delphi-faq.ru/palitra-komponentov-delphi/vizyalnie-biblioteki-vcl/svojstva-i-xarakteristiki-formy-form-v-delfi-delphi.html;

2 Borland Delphi для начинающих – Работа с формой [Электронный ресурс]. – Режим доступа: http://www.codenet.ru/progr/delphi/learn/workwithform.php;

3 Функция Sleep в Delphi [Электронный ресурс]. – Режим доступа: http://serj.kz/content/21.

3 Лабораторная работа №3. Прямая работа с файлами в среде Delphi

3.1 Цель работы

Изучить основные принципы работы с процедурами записи и чтения данных из файла в среде визуального программирования Delphi.

3.2 Ход работы

Задание общего уровня

1 Изучить методические рекомендации к лабораторной работе;

2 Создать новый проект delphi (запустить приложение delphi или выполнить команду file->new->application);

3 Спроектировать форму delphi, согласно рисунка 13, представленного в методических рекомендациях;

4 Организовать событие onclick для объекта button1;

5 По щелчку на кнопку организовать запись в текстовый файл тех вводимых чисел, которые являются четными, имя текстового файла любое (см. методические рекомендации);

6 Сохранить проект на внешнем носителе в директории с именем lab3_1;

7 Осуществить отладку и компиляцию проекта (клавиша f9);

8 В отчет занести постановку задачи, листинг программно модуля unit unit1, скриншот окна результата работ проекта и содержимое текстового файла;

9 В имеющемся проекте организовать метод, осуществляющий чтение записанных чисел в файле и подсчет среднего арифметического числа. Результат подсчета вывести на форму в компонент label (см. методические рекомендации);

10 Оформить форму, подписать и аккуратно расположить все компоненты;

11 Сохранить проект;

12 Осуществить отладку и компиляцию проекта;

13 Оформить отчет;

14 Преобразовать форму имеющегося проекта в соответствии с рисунком 14;

15 Для объекта button2 организовать событие onclick;

16 В локальном разделе var образовавшегося метода объявить переменные srar (среднее арифметическое) типа real, sum типа integer, i типа integer;

17 В образовавшемся методе организовать чтение чисел из созданного текстового файла (см. методические рекомендации);

18 Оформить форму, подписать и аккуратно расположить все компоненты;

19 Сохранить проект;

20 Осуществить отладку и компиляцию проекта;

21 Оформить отчет;

22 В уже имеющемся проекте организовать проверку существования файла, используя процедуры fileexist, ioresult;

Задание повышенного уровня

23 Создать новый проект delphi, выполнить индивидуальное задание;

24 Сохранить проект в директории с именем lab3_2;

25 Осуществить отладку и компиляцию проекта;

26 Оформить отчет, в который поместить постановку задачи индивидуального задания, листинг программного модуля и результат работы проекта;

27 Защитить лабораторную работу.

3.3 Содержание отчета

1 Цель работы;

2 Ход работы;

3 Постановка задачи на запись данных в файл, постановка задачи на чтение данных из файла;

4 Листинг программного модуля;

5 Результат работы приложение на запись данных в файл, содержимое файла, результат работы программы на чтение данных из файла.

3.4 Контрольные вопросы

1 Основные понятия объектно-ориентированного программирования: объект, свойство, событие, метод, класс. Привести программные примеры обращения или описания основных понятий;

2 Назовите основные принципы объектно-ориентированного программирования, приведите программные примеры;

3 Приведите примеры объявления файловой переменной;

4 Перечислите последовательность процедур, используемых для записи данных в файл;

5 Перечислите последовательность процедур для чтения данных из файла;

6 Какие процедуры используются для проверки существования файла, как можно обработать ошибку открытия файла в проекте.

3.5 Методические рекомендации

Технология работы с файлами в системе Delphi требует определённого порядка действий:

1 Файл должен быть открыт. Система следит, чтобы другие приложения не мешали работе с файлом. При этом определяется, в каком режиме открывается файл - для изменения или только считывания информации. После открытия файла в программу возвращается его идентификатор, который будет использоваться для указания на этот файл во всех процедурах обработки;

2 Начинается работа с файлом. Это могут быть запись, считывание, поиск и другие операции;

3 Файл закрывается. Теперь он опять доступен другим приложениям без ограничений. Закрытие файла гарантирует, что все внесённые изменения будут сохранены, так как для увеличения скорости работы изменения предварительно сохраняются в специальных буферах операционной системы.

В Delphi реализовано несколько способов работы с файлами. Прямая работа с файлами осуществляется через процедуры языка Pascal, связанными с использованием файловых переменных. Файловая переменная вводится для указания на файл.

var F: File;

Описанная таким образом файловая переменная считается *нетипизированной*, и позволяет работать с файлами с неизвестной структурой. Данные считываются и записываются побайтно блоками, размер которых указывается при открытии файла, вплоть от 1 байт.

Но чаще используются файлы, состоящие из последовательности одинаковых записей. Для описания такого файла к предыдущему описанию добавляется указание типа записи:

var F: File ofmun_zapuci;

В качестве типа могут использоваться базовые типы, или создаваться свои. Важно только, чтобы для типа был точно известен фиксированный размер в байтах,

поэтому тип String в чистом виде применяться не может.

Для обозначения файла, содержащего разноплановый набор символов используется следующие объявление:

var F: TextFile;

3.5.1 Открытие файла

Для открытия файла нужно указать, где он расположен. Для этого файловая переменная должна быть ассоциирована с нужным файлом, который определяется его адресом. Адрес файла может быть абсолютным, с указанием диска и каталогов ('C:\Мои документы\Мои рисунки\FileName.ini'), или относительным, тогда он создаётся в папке с .exe файлом программы. Для задания относительного адреса достаточно указать имя файла с нужным расширением. Делается это оператором AssignFile :

AssignFile(SaveF, 'C:\Мои документы\Мои рисунки\FileName.ini');

AssignFile(SaveF, 'FileName.ini');

Теперь файл должен быть открыт.

Открытие файла оператором **Rewrite** приведёт к воссозданию файла заново, т.е. существующий файл будет ***без предупреждения*** уничтожен, и на его месте будет создан новый пустой файл заданного типа, готовый к записи данных.Если же файла не было, то он будет создан.

Rewrite(SaveF);

Открытие файла оператором **Reset** откроет существующий файл к считыванию или записи данных, и его указатель будет установлен на начало файла:

Reset(SaveF);

Каждый из этих операторов может иметь второй необязательный параметр, имеющий смысл для нетипизированных файлов, и указывающий длину записи нетипизированного файла в байтах:

Rewrite(SaveF, 1);

Reset(SaveF, 1);

Открытие файла для добавления данных осуществляется оператором **Append**. Происходит добавление записей в конец файла.

Append(SaveF);

3.5.2 Чтение и запись данных файла

Чтение данных из файла можно осуществить, только, если он был открыт оператором **Reset.**

read(файловая_переменная, список_переменных)

readln(файловая_переменная, список_переменных)

Отличие между этими процедурами в том, что при вызове инструкции readln указатель чтения из файла автоматически перемещается в начало следующей строки файла.

Запись данных в файл осуществляется только в том случае, если файл открыт операторами **Rewrite** и **Append**.

Write(файловая_переменная, список_переменных)

Writeln(файловая_переменная, список_переменных)

Различие между инструкциями в том, что инструкция writeln после вывода всех значений, записывает в файл символ «новая строка».

При этом чтение и запись производится с текущей позиции указателя, затем указатель устанавливается на следующую запись. Можно проверить, существует ли нужный файл, оператором FileExists :

If FileExists('FileName.ini')thenRead(SaveF, SaveV);

Принудительно установить указатель на нужную запись можно оператором Seek(SaveF, N), где N - номер нужной записи, который, как и почти всё в программировании, отсчитывается от нуля:

Seek(SaveF, 49); - установка указателя на 50-ю запись.

При последовательном чтении из файла рано или поздно будет достигнут

конец файла, и при дальнейшем чтении произойдёт ошибка. Проверить, не достигнут ли конец файла, можно оператором EOF (аббревиатура EndOfFile), который равен true, если указатель установлен в конец файла:

while (not EOF(SaveF)) do

Read(SaveF, SaveV);

Оператор Truncate(SaveF) позволяет отсечь (стереть или удалить!) все записи файла, начиная от текущей позиции указателя, и до конца файла.

В конце работы с файлом его необходимо закрыть. Это делается оператором CloseFile(SaveF) ;

3.5.3 Ошибки открытия файла

Попытка открыть файл может завершиться неудачей и вызвать ошибку выполнения программы. Причин неудачи открытия файлов может быть несколько: программа может попытаться открыть файл на гибком диске, который не готов к работе или отсутствие открываемого в режиме добавления файла (файла нет — добавлять некуда).

Если программа запущена из Delphi, то при возникновении ошибки во время открытия файла, возникает исключение и на экране появляется диалоговое окно с сообщением об ошибке.

Программа может взять на себя задачу контроля результата выполнения инструкции открытия файла. Сделать это можно, проверив значение функции IOResult (Input-OutputResult — результат ввода-вывода). Функция IOResult возвращает 0, если операция ввода-вывода завершилась успешно; в противном случае — не ноль. Чтобы программа могла проверить результат выполнения операции ввода-вывода, нужно разрешить ей это делать. Для этого надо перед инструкцией вызова процедуры открытия файла поместить директиву компилятору — строку {$I-}, которая запрещает автоматическую обработку ошибок ввода-вывода. После инструкции открытия файла следует поместить директиву {$I+},

восстанавливающую режим обработки ошибок ввода/вывода.

AssignFile (f , filename) ;

{$I-}

Append (f)

{$I +}

if IOResult<>0 then Rewrite (f)

3.5.4 Запись чисел с формы Delphi в текстовый файл

Для организации записи данных с формы Delphi используются компоненты, в которые можно осуществить текстовый ввод данных. Например, компонент Edit, класса TEdit, Memoкласс TMemo, MaskEdit класс TMaskEdit и т.п.

Организовать форму в новом проекте Delphi, согласно следующего вида (рисунок 13).

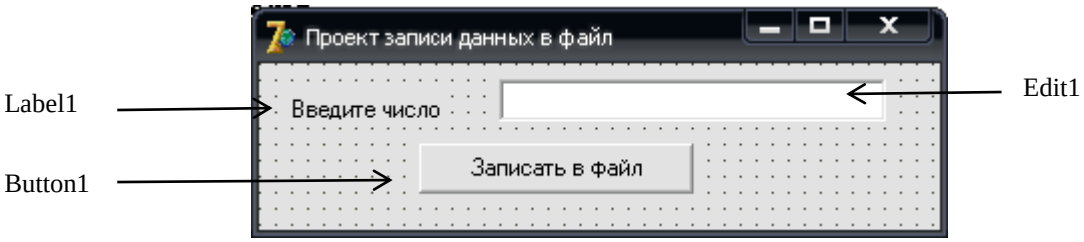


Рисунок 13 – Форма проекта в Delphi

Запись данных в текстовый файл организовать по щелчку. Ниже представлен обработчик события OnClick компонента Button1.

```
Procedure TForm1.Button1Click (Sender:TObject);
Var
   F:TextFile;
a:integer;
Begin
```

AssignFile (F,**'E:\1.txt'**); *//связываем переменную с физическим файлом, имя файла задать свое*

Append (F);*//открыть файл для добавления*

a:=StrToInt(Edit1.Text); //*переводим строку символов в число*

if a mod 2=0 //*если остаток от деления на 2 равен нулю, значит число четное – записываем его в файл*

then

begin

writeln(F,a); // *в файл записываем значение переменной a*

ShowMessage ('Число записано в файл!')

end

else

ShowMessage(‘Числонечетное!’);

Closefile(F); // *закрытьфайл*

End;

3.5.5 Чтение чисел из текстового файла, подсчет среднего арифметического значения

Для осуществления чтения данных из файла на форму Delphi необходимо добавить дополнительные компоненты (рисунок 14).

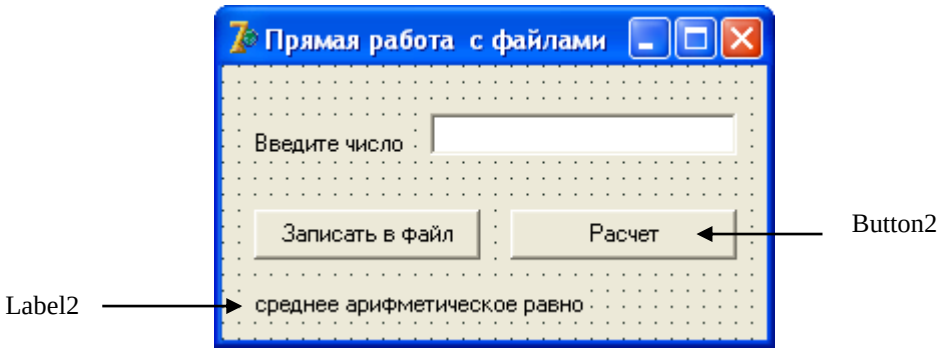


Рисунок 14 – Измененная форма проекта

Тело метода, осуществляющего чтение чисел из файла и расчет их среднего арифметического значения представлено ниже:

```
sum:=0;                  // в переменную суммы заносим пустое значение
i:=0;                    // «обнуляем» счетчик
AssignFile(F,'F:\1.txt'); // связываем файл с файловой переменной
Reset(F);                //открыть файл для чтения
WhilenotEof(f) do        // «пока не достигнут конец файла делать…»
begin
readln(F,a);             // читаем число в переменную «а»
sum:=sum+a;              // вычисляем сумму
i:=i+1;                  // счетчик количества чисел
end;                     // закрыть цикл «пока»
SrAr:=sum/i;             // расчет среднего арифметического
Label2.Caption:='Среднееарифметическоеравно - '+FloatToStr(SrAr);
CloseFile(F);            // закрыть файл
```

3.6 Варианты индивидуальных заданий

1 Составить программу, с помощью которой можно просматривать и редактировать список группы и информацию о каждом студенте. Форма может иметь вид, представленный на рисунке 15:

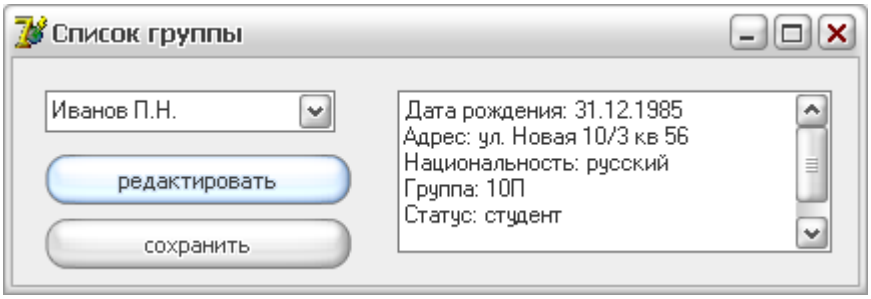


Рисунок 15 – Примерный вид формы

2 Разработать программу, которая в поле Memo выводит содержимое текстового файла. Рекомендуемый вид формы приведен на рисунке 16:

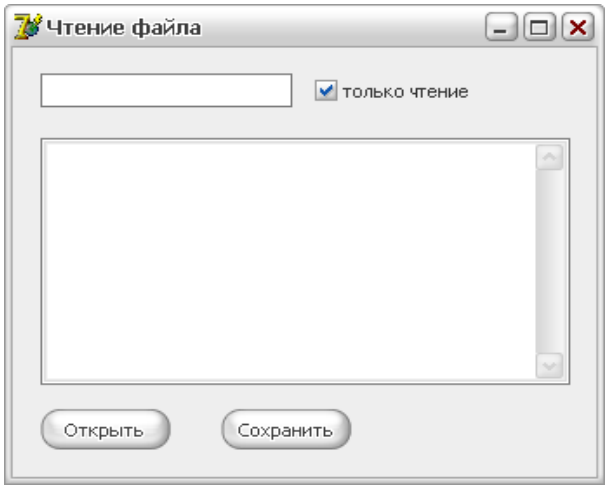


Рисунок 16 – Примерный вид формы

3 Разработать программу, которая в поле Memo выводит содержимое текстового файла. Для получения от пользователя имени файла используйте

стандартное диалоговое окно «Открытие файла». Рекомендуемый вид на рисунке 17:

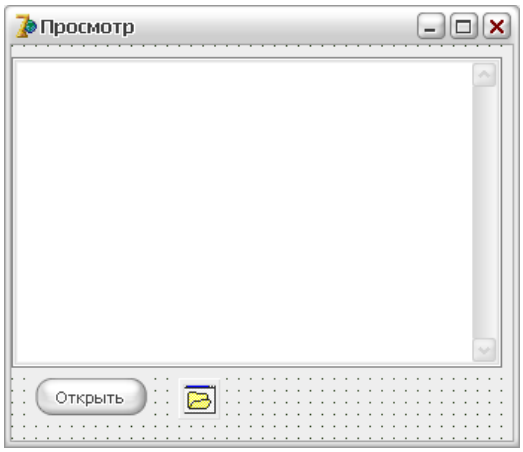


Рисунок 17 – Примерный вид формы

4 Разработать программу, которая добавляет ведет учет температуры воздуха в определенный день, реализованную в виде текстового файла, информацию о дневной температуре. Для ввода даты используйте компонент MonthCalendar. Если файл данных отсутствует, то программа должна его создать. Рекомендуемый вид на рисунке 18:

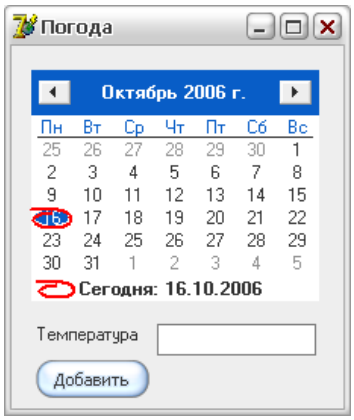


Рисунок 18 – Примерный вид формы

5 Разработать программу-ежедневник, которая добавляет в файл планы на текущий или предстоящий день. Рекомендуемый вид формы на рисунке 19:

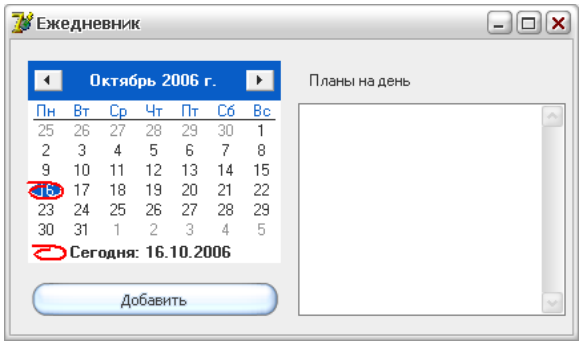


Рисунок 19 – Примерный вид формы

6 Разработать программу тестирования, в которой выбор правильного ответа осуществляется при помощи переключателей. Вопросы и варианты заданий хранятся в файле. В тесте задаются три вопроса: что такое инкапсуляция, наследование, полиморфизм. Рекомендуемый вид формы на рисунке 20:

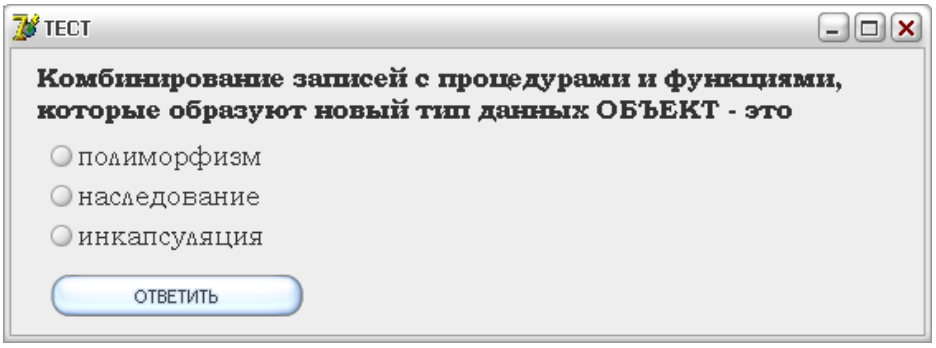


Рисунок 20 – Примерный вид формы

7 Разработать программу, которая ведет учет основных сведений о руководстве фирмы. Программа должна позволять вносить изменения и сохранять новые данные. Рекомендуемый вид формы на рисунке 21:

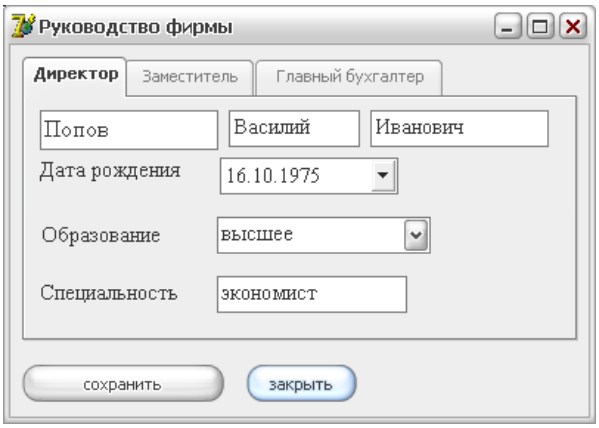


Рисунок 21 – Примерный вид формы

8 Разработать программу, которая выводит сведения о конфигурации ПК, характеристику устройства. Внешний вид формы аналогичен заданию №1;

9 Разработать программу, которая позволяет просматривать и корректировать сведения об организации (Название, полное именование, ИНН, Банк и его реквизиты, партнеры, список учредителей и т.п.). Рекомендуемый внешний вид аналогичен заданию №7;

10 Разработать программу «Ведомость успеваемости». Программа, должна содержать список группы и успеваемость конкретного студента по трем предметам. Рекомендуемый вид формы на рисунке 22:

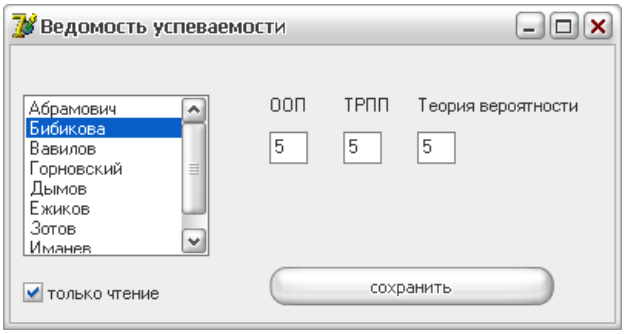


Рисунок 22 – Примерный вид формы

11 Разработать программу, которая позволяет оформлять заказ клиента фирмы и сохранять эту информацию в файле с его именем. Рекомендуемый вид формы на рисунке 23:

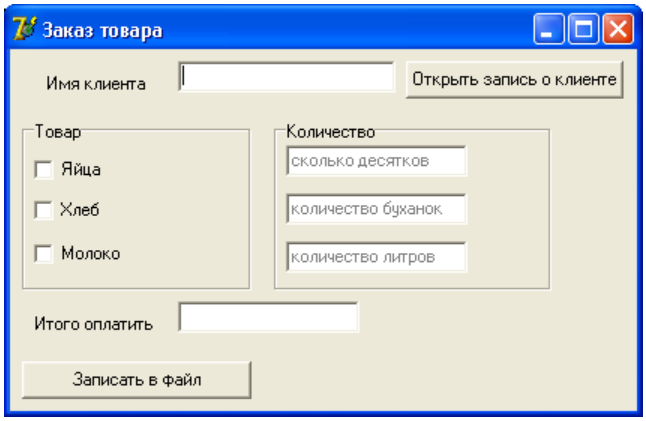


Рисунок 23 – Примерный вид формы

12 Разработать программу, которая позволяет просматривать список книг в библиотеке и редактировать его по желанию. Если книга удалена или получена новая, должен перезаписываться соответствующий файл;

13 Разработать программу, которая позволяет производить вычисления в матрице. Данные матрицы хранятся в файле. Программа производит считывание из файла и формирует их на форме. Рекомендуемый вид формы на рисунке 24:

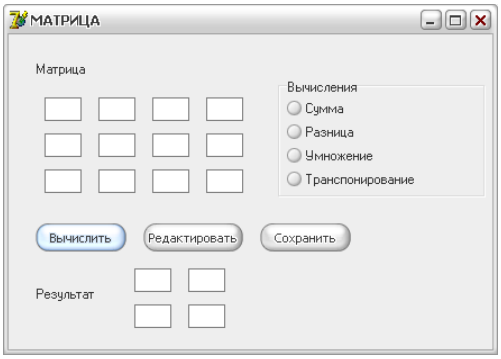


Рисунок 24 – Рекомендуемый внешний вид формы

14 Разработать программу, которая осуществляет ведение сведений о группах колледжа, их численности, кураторе и закрепленной аудиторией. Программа позволяет просматривать и редактировать данные. Внешний вид аналогичен рисунку 25:

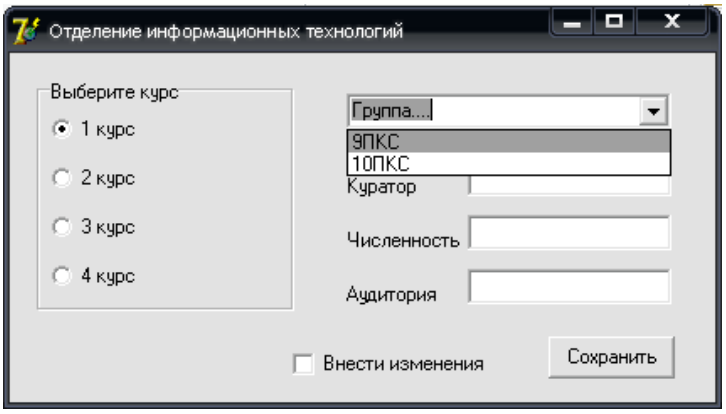


Рисунок 25 – Рекомендуемый внешний вид формы

15 Разработать программу, которая осуществляет хранение в файле данных о книгах в книжном магазине. В файл записывать сведения о книге, авторе, стоимости книги. Программа должна запрашивать у покупателя выбор книги, сумму за книгу, подсчитывать сдачу (или выводить сообщение о разнице в цене). Рекомендуемый вид формы на рисунке 26:

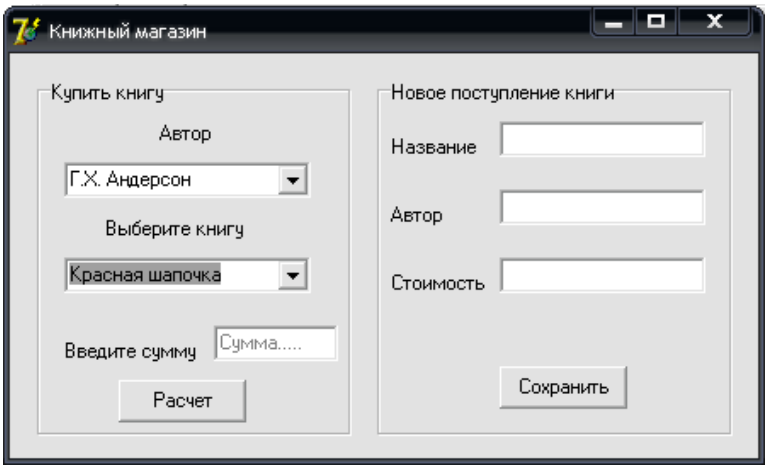


Рисунок 26 – Рекомендуемый внешний вид формы

4 Лабораторная работа №4. Динамическое распределение памяти

4.1 Цель работы

Научиться распределять память для экземпляров объекта, динамически создавать компоненты.

4.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Создать новый проект delphi, осуществляющий занесение данных об автопарке организации в список;

3 Спроектировать форму приложения, согласно рисунка 27, представленного в методических рекомендациях;

4 Описать родительский класс *tts* в секции *interface* программного модуля ниже описания класса *TForm1*;

5 В классе описать поля *fname* типа *string*, *fmosh_dvig* типа *integer*;

6 Описать конструктор *create*, осуществляющий присвоение полям объекта начальных значений;

7 Описать деконструктор *destroy*, осуществляющий уничтожение объекта.

8 Описать дочерний класс *tbus*;

9 В классе описать поле *fcount_places* типа *integer*;

10 Описать конструктор *create*, осуществляющий присвоение полям объекта начального значения;

11 Описать деконструктор *destroy*, осуществляющий уничтожение объекта;

12 Описать метод show для вывода списка автобусов;

13 В секции реализации implementation описать реализацию всех методов созданных классов (см. методические рекомендации);

14 Объявить секцию const после описания всех классов, задать переменную *count*, определяющую максимальное количество автобусов в списке;

15 В глобальном разделе описания переменных объявить массив автобусов *bus* типа *tbus*;

16 Объявить переменную «счетчик» количества автобусов;

17 Для компонента *button1* организовать событие *onclick*, осуществляющее создание списка;

18 Для компонента *button2* организовать событие *onclick*, осуществляющее вывод списка в компонент *memo1*;

19 Обработать дополнительные методы проекта;

20 Сохранить проект на диске в директории lab4_1;

21 Провести отладку и компиляцию проекта;

22 Реализовать созданные методы, при закрытии проекта вызвать их на выполнение;

23 Сохранить проект, выполнить отладку и компиляцию;

24 Создать новый проект delphi, демонстрирующий динамическое создание объекта *tmemo* на форме (см. методические указания п. 4.5.2);

25 Провести отладку и компиляцию проекта;

26 Сохранить проект на диске в директории lab4_2;

27 Оформить отчет о проделанной работе;

Задание повышенного уровня:

28 Выполнить индивидуальное задание согласно выданного варианта;

29 В проекте описать собственные классы, организовать методы, описать и реализовать конструкторы и деструкторы для объектов описанных классов;

30 Сохранить проект на диске в директории lab4_3;

31 Выполнить отладку и компиляцию проекта;

32 Оформить отчет о проделанной работе;

33 Защитить работу.

4.3 Содержание отчета

1 Цель, ход работы;

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

3 Постановка задачи на создание динамического компонента delphi, листинг программного модуля, результат работы приложения до создания объекта и после;

4 Формулировка индивидуального задания;

5 Листинг программного модуля индивидуального задания, результат работы приложения;

6 Вывод.

4.4 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 Каковы функции использования конструкторов и деструкторов?

5 Правила объявления конструкторов и наследуемых конструкторов;

6 Для чего используется директива inherited?

7 Какова функция указателя self?

4.5 Методические рекомендации

4.5.1 Понятие конструктора и деструктора

Методы, которые предназначены для создания и удаления объектов называются конструкторами и деструкторами соответственно. Описание данных методов отличается от обычных тем, что в их заголовках стоят ключевые слова *constructor* и *destructor*. В качестве имен конструкторов и деструкторов в базовом классе *TObject* и многих других классах используются имена *Create* и *Destroy*.

Конструкторы и деструкторы отвечают за существование объекта в памяти, т.е. выделяют память для экземпляра класса, затем и освобождают ее.

Конструктор – это специальный вид подпрограммы, присоединенный к классу. Его назначение – создавать представителей (экземпляры) класса. Он ведет себя как функция, которая возвращает ссылку на вновь созданный экземпляр класса, т.е. на объект. Одновременно выделяется память для хранения значений полей экземпляра класса.

Деструктор – это специальная разновидность подпрограммы, присоединенной к классу. Его назначение заключается в уничтожении экземпляра класса, т.е. объекта и освобождении памяти, выделенной под экземпляр.

Синтаксис объявления конструкторов и деструкторов:

Type

*<имя класса>=**Class**[{Имя родительского класса>)]*

Constructor *<Имя конструктора>[(<параметры>)]; [**Override**;]*

Destructor *<Имя деструктора>[(<параметры>)}; [**Override**;]*

End;

Примечания:

1 Объявляются конструкторы и деструкторы, как правило, в разделе *Public* класса;

2 В классе может быть объявлено несколько конструкторов, однако чаще бывает один конструктор. Общепринятое имя для единственного конструктора *Create*;

3 В одном классе может быть объявлено несколько деструкторов, но чаще бывает один деструктор без параметров с именем *Destroy*;

4 За объявлением деструктора по имени *Destroy* следует указывать ключевое слово-директиву *Override*, разрешающее выполнение предусмотренных по умолчанию действий для уничтожения экземпляра объекта, если при его создании возникла какая-либо ошибка. Фактически *Override* переопределяет метод предка;

5 Метод *Free* так же удаляет (разрушает) экземпляры класса, предварительно проверяя их на *Nil*.

Реализация конструктора

В задачу конструктора входит создание экземпляра класса и выполнение операторов, содержащихся в его теле. Назначение кода внутри конструктора – инициализировать только что созданный экземпляр объекта. Синтаксис реализации конструктора:

Constructor *<имя класса>.<имя конструктора>[(<параметры>}};*

[<блок объявлений>}

Begin

<Исполняемые операторы>

End*;*

Реализация наследуемых конструкторов.

Constructor *<имя класса>.<имя конструктора>[(<параметры>)];*

[<блок обьявлений>}

Begin

Inherited *<имя конструктора>[(<параметры>)};*

<инициализация собственных полей>

End*;*

Реализация деструкторов

Деструктор уничтожает экземпляр класса, который был использован при его

вызове, автоматически освобождая любую динамическую память, которая ранее была зарезервирована конструктором, закрывает файлы и т.п. операции. Программист ответственен за вызов деструкторов для всех экземпляров класса, если были зарезервированы подчиненные объекты. Часто деструктор не выполняет других действий и представляет собой пустую процедуру. Синтаксис реализации деструктора:

Destructor *<имя класса>.<имя деструктора>[(<параметры>)];*

[<блок объявлений>]

Begin

<исполняемые операторы>

End;

Реализация наследуемых деструкторов

Если использовать механизм наследования деструкторов, то можно упростить задачу уничтожения экземпляров класса, таким образом, чтобы каждый раз заботиться лишь об уничтожении тех полей, которые были добавлены в данном классе. Всю работу по очистке наследуемых полей можно возложить на наследуемые деструкторы. Для вызова наследуемого деструктора необходимо используется ключевое слово *Inherited*.

Синтаксис объявления наследуемого деструктора следующий:

Destructor *<имя класса>.<имя деструктора>[(<параметры>)};*

[<блок объявлений>]

Begin

<уничтожение собственных полей>

Inherited *<имя деструктора>[{<параметры>)];*

End;

Вызов конструкторов

Для того чтобы вызвать конструктор, необходимо правильно объявить и определить класс. Объявление класса должно быть доступно, т.е. он должен находится в области видимости из того места, где конструктор будет вызываться.

Если в пользовательском классе не определен конструктор, то по умолчанию

будет использоваться конструктор, унаследованный от класса-потомка. В любом случае у всех объектов есть доступ к конструктору *Create*, определенному в классе *TObject*.

Определение класса создает активную структуру, способную создавать представителей этого класса. Объекты, которые создаются с помощью определения класса, способны хранить ссылки на вновь создаваемые объекты.

Var *<имя обьекта>: <имя класса>; // Объявление переменной–указателя*

Begin

<имя обьекта>:=<имя класса>.<имя конструктора>[(<параметры>)];

Если вызвать конструктор от имени объекта, то новый объект не будет создан (память не выделяется), но будут выполнены операторы, указанные в коде конструктора. Конструктор может также вызываться с помощью переменной типа указателя на класс.

Вызов деструкторов

Деструкторы вызываются точно так же, как и большинство других методов класса – через его действующий экземпляр. Синтаксис вызова конструктора следующий:

<имя объекта>.<имя деструктора>[(<параметры>)];

Для примера опишем родительский класс транспортных средств *TTs*, дочерний класс *Автобусы* (*TBus*).

Интерфейс разрабатываемого приложения может иметь вид, представленный на рисунке 27.

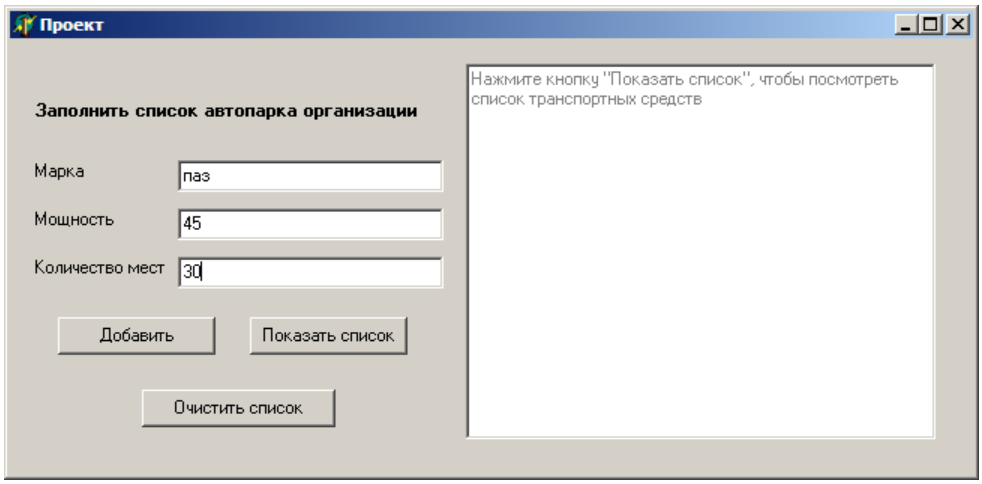


Рисунок 27 – Интерфейс приложения

В разделе Type определим родительский класс *TTs*. Класс содержит поля *наименование* (*fname*) и *мощность двигателя (fmosh_dvig)*, конструктор *Create*.

```
type TTs = class
  fname:string[20];
  fmosh_dvig: integer;
 constructor Create (name:string; mosh_dvig:integer);
 destructor Destroy;
end;
```

Определим дочерний класс *TBus*. Класс содержит поля (*count_places*), конструктор *Create*, метод *Show* для вывода информации об автобусах в компонент *TMemo*.

```
TBus = class (TTs)
fcount_places: integer;
constructor Create (name:string; mosh_dvig:integer; count_places:integer);
destructor Destroy;
procedure show (Mem:TMemo);
end;
```

В разделе реализации определяем конструктор класса *TTs*.

```
constructor TTs.Create (name:string; mosh_dvig:integer);
begin
  fname:=name;
  fmosh_dvig:=mosh_dvig;
end;
```

Реализация конструктора класса *TBus* выглядит следующим образом.

```
constructor TBus.Create (name:string; mosh_dvig:integer; count_places:integer);
begin
  inherited create (name, mosh_dvig);
  fcount_places:=count_places;
end;
```

Реализация деструкторов классов *TTs* и *TBus* соответственно.

```
destructor TTs.Destroy;
begin
end;
destructor TBus.Destroy;
begin
  inherited;
end;
```

Реализация метода *Show* класса *TBus*.

```
procedure tbus.show (Mem:TMemo);
begin
  mem.lines.add(IntToStr(i)+')'+ fname+' '+'Мощность двигателя
='+IntToStr(fmosh_dvig)+' л.с.'+' Кол-во посад.мест= '+IntToStr(fcount_places));
end;
```

В разделе объявления констант *const* (данный раздел располагается между разделом описания типов и разделом переменных) объявляем значение константы *count* – максимальное число элементов массива (списка автобусов).

```
const count=5;
```

В разделе переменных *Var* объявляем массив объектов (список автобусов) *Bus* класса *TBus*.

```
var
  Form1: TForm1;
  Bus:array [1..count] of TBus;
  i,n:integer; //счетчики элементов массива Bus
```

Обрабатываем событие *OnCreate* для формы *Form1*. При создании формы переменной (счетчику) *n* присваивается значение равное *1*.

```
procedure TForm1.FormShow(Sender: TObject);
begin
   n:=1;
end;
```

Обработчик события *OnClick* для объекта *Button1*. Осуществляет создание объектов класса *TBus*, помещает данные об автобусах в список.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 if  n<=count then
 begin
     Bus[n]:=TBus.Create(edit1.Text,StrToInt(edit2.Text),StrToInt(edit3.text));
     n:=n+1;
 end
else ShowMessage('Список заполнен!');
   edit1.Text:='';
   edit2.Text:='';
   edit3.Text:='';
end;
```

Вывод списка осуществляется применением метода *show* к элементам массива (обработчик события *OnClick* для объекта *Button2*):

```
procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
  memo1.Lines.Clear;
  for i:=1 to count do
  if Bus[i] <> NIL then Bus[i].show(Memo1);
end;
```

Обработчик события *OnClick* для объекта *Button3*. Осуществляет уничтожение объектов класса *TBus*

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  for i:=1 to count do
  if Bus[i]<>NIL then
  begin
     Bus[i].Destroy;
     Bus[i]:=nil;
  end;
  memo1.Lines.Clear;
  n:=1;
end;
```

4.5.2 Динамическое создание компонентов

Вызовы конструкторов и деструкторов визуальных компонентов Delphi. Любой компонент, попавший приложение при визуальном проектировании, включается в определенную иерархию объектов, которая замыкается на форме (класс *TForm*). Поэтому вызов конструкторов и деструкторов всех компонентов формы производится автоматически при инициализации и удалении формы, незримо для программиста.

Сами формы создаются и уничтожаются приложением – глобальным объектом с именем *Application*. В файле-проекте с расширением **.dpr* можно увидеть вызов конструктора формы в виде строки:

Application.CreateForm(Tform1, Form1);

Динамически создаваемые компоненты – это компоненты, место в памяти под которые выделяется по мере необходимости в процессе работы приложения. Этим они и отличаются от компонентов, которые помещаются на Форму при проектировании приложения. Возможность создавать компоненты динамически это очень большое удобство для программиста. Например, можно создавать в цикле сразу много однотипных компонентов, формируя из них массив, которым в дальнейшем очень просто управлять.

Все компоненты, как объекты, имеют множество свойств, определяющих их работу. При установке компонента на Форму из палитры большинство этих свойств определяются системой Delphi автоматически. При создании динамического компонента программист должен описать и настроить их вручную.

Прежде всего, для появления динамически создаваемого компонента нужно выделить под него место в памяти. Выделением места в памяти компьютера под любой компонент занимается конструктор типа объекта этого компонента – метод *Create*. Для этого сначала нужно описать переменную нужного типа, а затем для выделения памяти воспользоваться методом *Create*. Метод *Create* имеет параметр *Owner*, определяющий так называемого «владельца» для создаваемого компонента.

При обычной установке компонента из палитры система делает владельцем этого компонента Форму. Проще всего поступать так же. Однако можно указать в качестве владельца сам этот компонент, воспользовавшись в качестве параметра ключевым словом *Self*.

Когда компонент создан, то есть место в памяти под него выделено, можно задавать значения параметрам этого объекта. Прежде всего, это ещё один компонент, так называемый «родитель». Компонент-родитель будет отвечать за отрисовку нашего динамически создаваемого компонента. Это значит, что новый компонент появится в границах компонента-родителя.

Если компонент-владелец имеет тип *Tcomponent*, то есть может быть любым компонентом, то компонент-родитель уже имеет тип *TwinControl*. То есть это

должен быть «оконный» компонент, умеющий принимать и обрабатывать сообщения от системы Windows. Это необходимо, так как компонент должен находиться в некоторой иерархии компонентов, принимающих и передающих сообщения от системы Windows. Нашему динамическому компоненту сообщения будут передаваться через компонент-родитель.

Компонент не может быть родителем для самого себя. Имя компонента-родителя просто присваивается свойству *Parent* создаваемого динамически компонента. Общая схема «конструирования» динамически создаваемого компонента:

```
var Component: Tcomponent;//Описать переменную для компонента
begin
  Component:=Tcomponent.Create(Owner);//Задать владельца
      Component.Parent:=Parent;//Задать родителя
  end;
```

На этом создание компонента можно считать законченным, и он успешно появляется (или «не появляется», если он не визуальный) в приложении. Остальные свойства будут присвоены ему по умолчанию самой системой Delphi.

Для примера динамически создадим многострочный редактор, компонент *Memo*. Пусть он появляется на форме по нажатию кнопки:

```
procedureTForm1.Button1Click(Sender: Tobject);
var Memo: Tmemo;
begin
Memo:=Tmemo.Create(Form1);
Memo.Parent:=Form1;
Memo.Left:=50;
Memo.Top:=50;
Memo.Width:=250;
Memo.Height:=100;
Memo.Text:='Компонент Memo динамически создан!';
end;
```

По умолчанию Delphi присвоит ему типовое имя с присвоением очередного порядкового номера: *Memo1*. Программист при создании компонента также может присвоить свойству *Name* нужное значение, например:

Memo.Name:='DynamMemo';

К данному компоненту можно обращаться как по этому имени, так и с указанием переменной, с помощью которой он был создан: *Memo*. В последнем случае переменная должна быть глобальной.

Создадим кнопку удаления динамически созданного объекта Memo. Формируем обработчик события *OnClick*.

If Memo<>nil then a.Destroy;

4.6 Варианты индивидуальных заданий

1 Сформировать список городов, объявив в качестве родительского класс Tterritoria;

2 Сформировать список отделений больницы, объявив в качестве родительского класс Thospital;

3 Сформировать список молочных продуктов, объявив в качестве родительского класс Tproduct;

4 Сформировать список кафедр, объявив в качестве родительского класс Tpodrazdel;

5 Сформировать список ноутбуков, объявив в качестве родительского класс Tcomputer;

6 Сформировать список журналов, объявив в качестве родительского класс Tizdanie;

7 Сформировать список заводов Оренбургской области, объявив в качестве родительского класс Tenterprise;

8 Сформировать список программистов, объявив в качестве родительского класс Tsotrudnik;

9 Сформировать список крылатых насекомых, объявив в качестве родительского класс Tnasekom;

10 Сформировать список служебных программ, объявив в качестве родительского класс Tpo;

11 Сформировать список квартир, объявив в качестве родительского класс Tnedvizh;

12 Сформировать список озер Оренбургской области, объявив в качестве родительского класс Tvodoem;

13 Сформировать список фигуристов, объявив в качестве родительского класс Tsportsman;

14 Сформировать список рекламных агентств, объявив в качестве родительского класс Tservice;

15 Сформировать список радиостанций, объявив в качестве родительского класс Tsmi.

5 Лабораторная работа №5. Создание виртуального метода в классе

5.1 Цель работы

Научиться описывать классы, содержащие виртуальные методы, согласно правил вызова.

5.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Создать новый проект delphi, осуществляющий занесение данных о

животных зоопарка в список;

3 Спроектировать форму приложения, согласно рисунка 29, представленного в методических рекомендациях;

4 Описать родительский класс tanimal в секции interface программного модуля ниже описания класса tform1:

а) в классе описать поле NameAnimal типа string;

б) описать конструктор Create, осуществляющий присвоение полю объекта начального значения;

в) описать метод GetInfo, который будет формировать строки списка, метод сделать виртуальным;

5 Описать дочерний класс Tmammal:

а) в классе описать поле age типа real;

б) описать конструктор Create, осуществляющий присвоение полям объекта начального значения;

в) описать метод GetInfo, согласно всех правил вызова виртуальных методов;

6 Описать дочерний класс Treptiles:

а) в классе описать поле Length типа real;

б) описать конструктор Create, осуществляющий присвоение полям объекта начального значения;

в) описать метод GetInfo, согласно всех правил вызова виртуальных методов;

7 В секции реализации implementation описать реализацию всех методов трех созданных классов (см. методические рекомендации);

8 Объявить секцию const после описания всех классов, задать переменную, определяющую максимальное количество животных в списке;

9 В глобальном разделе описания переменных объявить массив животных типа tanimal;

10 Объявить переменную «счетчик» количества животных;

11 Для компонента button1 организовать событие onclick, осуществляющее

создание списка;

12 Для компонента button2 организовать событие onclick, осуществляющее вывод списка на экран;

13 Обработать дополнительные методы проекта;

14 Сохранить проект на диске в директории lab5_1;

15 Провести отладку и компиляцию проекта;

16 Дописать деструктор в каждом из трех созданных классов;

17 Реализовать созданные методы, при закрытии проекта вызвать их на выполнение;

18 Сохранить проект, выполнить отладку и компиляцию;

19 Оформить отчет о проделанной работе;

Задание повышенного уровня:

20 Выполнить индивидуальное задание согласно выданного варианта;

21 В проекте описать собственные классы, организовать виртуальные методы, описать и реализовать конструкторы и деструкторы для объектов описанных классов;

22 Сохранить проект на диске в директории lab5_2;

23 Выполнить отладку и компиляцию проекта;

24 Оформить отчет о проделанной работе;

25 Защитить работу.

5.3 Содержание отчета

1 Цель, ход работы;

2 Постановка задачи №1, листинг программного модуля, результат работы приложения;

3 Формулировка индивидуального задания;

4 Листинг программного модуля индивидуального задания, результат работы приложения;

5 Вывод.

5.4 Контрольные вопросы

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

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

3 Что представляет собой раннее и позднее связывание?

4 В чем заключается смысл вызова виртуального метода, отличие виртуальных, динамических и статических методов;

5 Перечислите правила вызова виртуальных методов, особенности объявления виртуального метода в дочерних классах;

6 Каково назначение конструктора и деструктора?

7 Для чего используется директива inherited?

5.5 Методические рекомендации

5.5.1 Понятие полиморфизма. Раннее и позднее связывание

Полиморфизм – это свойство объектов разных классов выполнять одно и то же действие согласно своего типа.

Два или более класса, которые являются производными одного и того же базового класса, называются полиморфными. Это означает, что они могут иметь общие характеристики, но так же обладать собственными свойствами.

В рамках ООП поведенческие свойства объекта определяются набором входящих в него методов. Изменяя алгоритм того или иного метода в потомках объекта, программист может придавать этим потомкам отсутствующие у родителя

специфические свойства. Для изменения метода необходимо перекрыть его в потомке, т.е. объявить в потомке одноименный метод и реализовать в нем нужные действия. В результате чего в объекте-родителе и объекте-потомке будут действовать два одноименных метода, имеющих разную алгоритмическую основу и, следовательно, придающие объектам разные свойства. Это и называется полиморфизмом объектов.

```
Type
TAnimal=class
...
Procedure GetEat;
End;
TMammal=class(TAnimal)
...
Procedure GetEat(count:real);
End;
```

Методы объектов бывают статическими, виртуальными и динамическими.

Статические методы включаются в код программы при компиляции. Это означает, что до использования программы определено, какая процедура будет вызвана в данной точке. Компилятор определяет, какого типа объект используется при данном вызове, и подставляет метод этого объекта.

Объекты разных типов могут иметь одноименные статические методы. В этом случае нужный метод определяется по типу экземпляра объекта.

Это удобно, так как одинаковые по смыслу методы разных типов объектов можно и назвать одинаково, а это упрощает понимание и задачи и программы. Статическое перекрытие – первый шаг полиморфизма. Одинаковые имена – вопрос удобства программирования, а не принцип.

Процесс, с помощью которого вызовы статических методов однозначно разрешаются компилятором во время компиляции в один метод, называется **ранним связыванием**. При раннем связывании вызывающий и вызываемый методы связываются при первой же возможности, т.е. во время компиляции. При **позднем связывании** вызывающий и вызываемый методы не могут быть связаны во время

компиляции, поэтому включается механизм, позволяющий осуществить связывание несколько позднее, когда вызов действительно произойдет.

Виртуальные методы в отличие от статических, подключаются к основному коду на этапе выполнения программы. Виртуальные методы дают возможность определить тип и конкретизировать экземпляр объекта в процессе исполнения, а затем вызвать методы этого объекта.

Описание виртуального метода отличается от описания обычного метода добавлением после заголовка метода служебного слова **virtual** .

procedure Method (список параметров); ***virtual;***

Объявление виртуального метода в базовом классе выполняется с помощью ключевого слова **virtual**, а его перекрытие в производных классах - с помощью ключевого слова **override**. Перекрытый метод должен иметь точно такой же формат (список параметров, а для функций еще и тип возвращаемого значения), что и перекрываемый.

```
Type
TAnimal=class
...
Procedure GetEat;
Function GetInfo : string; virtual;
End;
TMammal=class(TAnimal)
...
Function GetInfo : string; override;
End;
```

Использование виртуальных методов в иерархии типов объектов имеет определенные ограничения:

1 Если метод объявлен как виртуальный, то в типе потомка его нельзя перекрыть статическим методом;

2 Объекты, имеющие виртуальные методы, инициализируются специальными процедурами, которые, в сущности, также являются виртуальными и носят название **constructor**;

3 Списки переменных, типы функций в заголовках перекрывающих друг друга виртуальных процедур и функций должны совпадать полностью;

Обычно на **конструктор** возлагается работа по инициализации экземпляра объекта: присвоение полям исходных значений, первоначальный вывод на экран и т.п.

Помимо действий, заложенных в него программистом, конструктор выполняет подготовку механизма позднего связывания виртуальных методов. Это означает, что еще до вызова любого виртуального метода должен быть выполнен какой-нибудь конструктор.

Конструктор – это специальный метод, который инициализирует объект, содержащий виртуальные методы. Заголовок конструктора выглядит так:

constructor Method (список параметров);

Зарезервированное слово **constructor** заменяет слова procedure и virtual .

Основное и особенное назначение конструктора – установление связей с таблицей виртуальных методов (VMT) – структурой, содержащей ссылки на виртуальные методы. Таким образом, конструктор инициализирует объект установкой связи между объектом и VMT с адресами кодов виртуальных методов. При инициализации и происходит позднее связывание.

У каждого объекта своя таблица виртуальных методов VMT . Именно это и позволяет одноименному методу вызывать различные процедуры.

Упомянув о конструкторе, следует сказать и о **деструкторе**. Его роль противоположна: выполнить действия, завершающие работу с объектом, закрыть все файлы, очистить динамическую память, очистить экран и т.д.

Заголовок деструктора выглядит таким образом:

destructor Done ;

Основное назначение деструкторов – уничтожение VMT данного объекта. Часто деструктор не выполняет других действий и представляет собой пустую процедуру.

```
destructor Done ;
begin
end ;
```

Пример описания корректного описания виртуального метода:

```
Type
TAnimal=class
…
Constructor Create(NameAnimal:string);
Function GetInfo : string; virtual;
End;
TMammal=class(TAnimal)
…
Constructor Create(NameAnimal:string; Age:real);
Function GetInfo : string; override;
End;
….
Var
AnyAnimal:TAnimal;
```

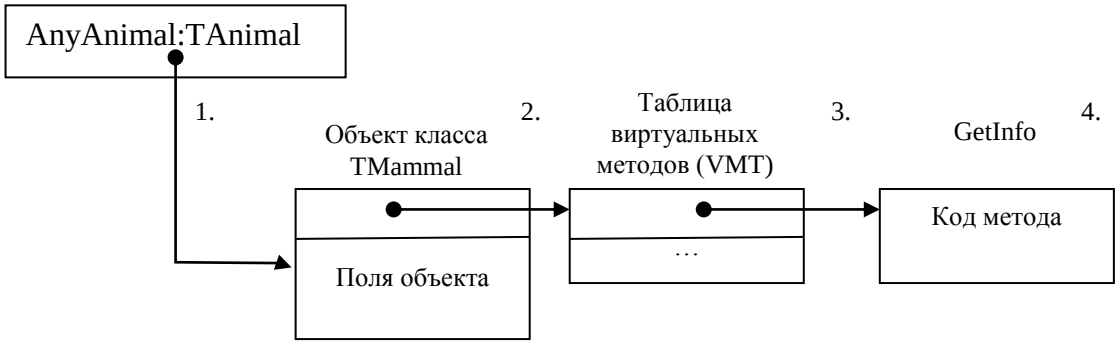


Рисунок 28 – Механизм вызова виртуального метода

Вызов виртуального метода осуществляется следующим образом:

1 Через объектную переменную выполняется обращение к занятому объектом блоку памяти;

2 Далее из этого блока извлекается адрес таблицы виртуальных методов (он записан в четырех первых байтах);

3 На основании порядкового номера виртуального метода извлекается адрес соответствующей подпрограммы;

4 Вызывается код, находящийся по этому адресу.

В реализации метода конструктора в дочернем классе используется директива **inherited**. Ключевое слово **Inherited** используется, чтобы вызвать родительский конструктор или метод деструктора, как соответствующий для текущего класса.

Оно вызывается в начале конструктора, и в конце деструктора. Это не является обязательным, но рекомендуется.

Без параметров Inherited вызывает так же названный метод родительского класса, с теми же самыми параметрами.

1 Create;

```
Begin
Inherited; // Всегда вызывается в начале конструктора
...
end;
```

2 Create(arguments);

```
Begin
Inherited Create(arguments); // Всегда вызывается в начале конструктора
...
end;
```

3 Destroy

```
Begin
...
Inherited; // Всегда вызывается в конце деструктора
```

end;

5.5.2 Задание для выполнения

Постановка задачи: Спроектировать приложение, в котором осуществить запись данных о животных зоопарка в список. Вывод списка животных на экран по щелчку на кнопку.

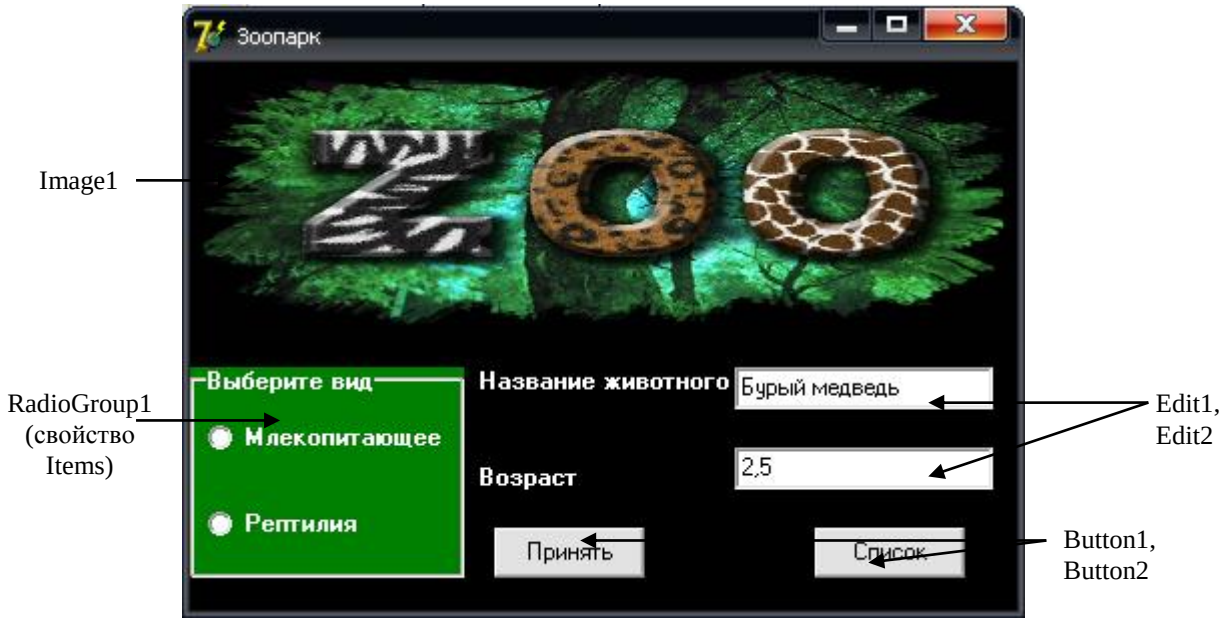


Рисунок 29 – Примерный внешний вид формы

Родительский класс:

type

TAnimal=class

NameAnimal:string;

constructor Create (FNameAnim:string) ;

function GetInfo: string; virtual;

end;

Дочерний класс (Млекопитающие)

TMammal=class(TAnimal)

Age:real;

```
constructor Create(FNameAnim:string; FAge:real);
function GetInfo:string; override;
end;
```

Дочерний класс (Рептилии)

```
TReptiles=class(TAnimal)
Length:real;
constructor Create(FNameAnim:string; FLength:real);
function GetInfo:string; override;
end;
```

Реализация методов класса TAnimal:

```
constructor TAnimal.Create (FNameAnim:string);
begin
NameAnimal:=FNameAnim;
end;
function TAnimal.GetInfo:string;
begin
Result:=NameAnimal;
end;
```

Реализация методов класса TMammal:

```
constructor TMammal.Create(FNameAnim:string; FAge:real);
begin
inherited create(FNameAnim);
Age:=FAge;
end;
function TMammal.GetInfo:string;
begin
result:=NameAnimal+' возраст ' +FloatToStr(Age);
end;
```

Реализация методов класса TReptiles:

```
constructor TReptiles.create(FNameAnim:string; FLength:real);
```

```
begin
inherited create(FNameAnim);
Length:=FLength;
end;
function TReptiles.GetInfo:string;
begin
result:=NameAnimal+' длина животного '+FloatToStr(Length);
end;
```

Раздел констант в среде Delphi располагается между секцией описания типов и глобальным разделом описания переменных

```
const
Size=10;  // размер списка
var
  Form1: TForm1;
  Massiv: array[1..Size] of TAnimal; // список
  n:integer;     // счетчик количества животных в списке
```

Обработчик события OnClick для объекта Button1. Осуществляет создание объектов описанных классов, помещает данные о животных в список.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
IF  n<=Size THEN
begin
if RadioGroup1.ItemIndex=0
then                                    // создадим объект TMammal
                Massiv[n]:=TMammal.Create(Edit1.Text,StrToFloat(Edit2.Text))
Else                                    // создадим объект TReptiles
Massiv[n]:=TReptiles.Create(Edit1.Text,StrToFloat(Edit2.Text));

n:=n+1;
End
```

ELSE ShowMessage('Список заполнен!');

end;

Обработчик события OnClick компонента Button2, осуществляющий вывод списка животных в окне сообщения.

```
procedure TForm1.Button2Click(Sender: TObject);
var
i:integer ;
st:string;
begin
for i:=1 to Size do
                    if list[i] <> NIL then st:=st+Massiv[i].Getinfo+#13;
ShowMessage('Животные зоопарка: '+#13+st);
end;
```

Дополнительные методы проекта

Задание начального значения элементу списка:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
n:=1;
end;
```

Формирование текста надписи в зависимости от выбранного животного:

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
if RadioGroup1.ItemIndex=0 then Label2.Caption:='Возраст';
if RadioGroup1.ItemIndex=1 then Label2.Caption:='Длина животного';
end;
```

5.6 Варианты индивидуальных заданий

1 Сформировать, применяя метод полиморфизма список обитателей озера Байкал, объявив родительский класс как TFauna;

2 Сформировать, применяя метод полиморфизма список обитателей с/х фермы, родительский класс TFerma;

3 Сформировать, применяя метод полиморфизма список лекарственных растений, родительский класс TFlora;

4 Сформировать, применяя метод полиморфизма список животных опасных для жизни человека, родительский класс TFauna;

5 Сформировать, применяя метод полиморфизма список ядовитых растений, родительский класс TFlora;

6 Сформировать, применяя метод полиморфизма список музыкальных инструментов для оркестра, родительский класс TInstument;

7 Сформировать, применяя метод полиморфизма список обитателей реки Лимпопо, родительский класс TFauna;

8 Сформировать, применяя метод полиморфизма список инструментов, применяемых в строительстве, родительский класс TInstrument;

9 Сформировать, применяя метод полиморфизма список цветов для создания композиции, родительский класс TFlora;

10 Сформировать, применяя метод полиморфизма список орудий Второй Мировой Войны, родительский класс TOrygie;

11 Сформировать, применяя метод полиморфизма список транспорта аэрофлота города, родительский класс TTransport;

12 Сформировать, применяя метод полиморфизма список восточных единоборств, родительский класс TMartialArts;

13 Сформировать, применяя метод полиморфизма список жанров кино, родительский класс TCinema;

14 Сформировать, применяя метод полиморфизма список уровней компьютерной игры, родительский класс TGame;

15 Сформировать, применяя метод полиморфизма список канцелярских принадлежностей школьника, родительский класс TStationery.

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

6.1 Цель работы

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

6.2 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 30, 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 31, представленного в методических рекомендациях:

а) на форме разместить компонент *TStringGrid*, отображающий исходные данные;

б) разместить компонент *TChart*; вызвать редактор диаграммы, щелкнув два раза по компоненту; установить необходимые параметры диаграммы (см. методические рекомендации п. 6.5.2);

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 32, представленного в методических рекомендациях:

а) на форме разместить компонент *TStringGrid*, отображающий исходные данные;

б) разместить компонент *TChart*; вызвать редактор диаграммы, щелкнув два раза по компоненту; установить необходимые параметры диаграммы (см. методические рекомендации п. 6.5.2);

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. 6.5.3.4;

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.6.6);

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.6.6);

27 Сохранить проект;

28 Провести отладку и компиляцию проекта;

29 Оформить отчет о проделанной работе;

30 Защитить работу.

6.3 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.

6.4 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*.

6.5 Методические рекомендации

6.5.1 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 7 перечислены некоторые свойства компонента *StringGrid*.

Таблица 7 – 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 8 перечислены некоторые события компонента *StringGrid*.

Таблица 8 – 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 30. 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.*

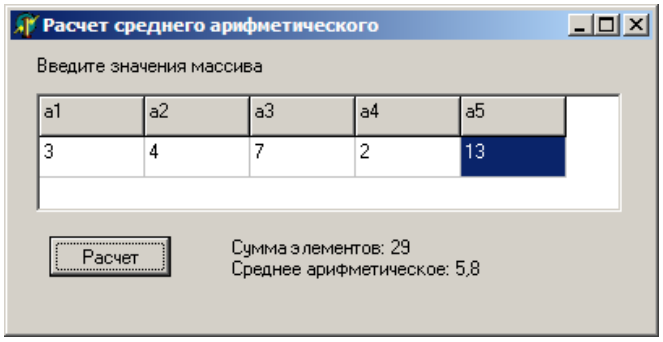


Рисунок 30 – Внешний интерфейс приложения

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;
```

6.5.2 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 9.

Таблица 9 – 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 9

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 31. 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*.

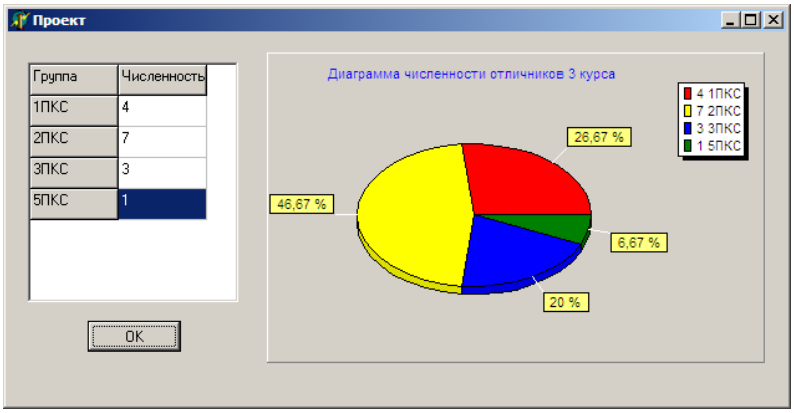


Рисунок 31 – Внешний интерфейс приложения

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 32.

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*.

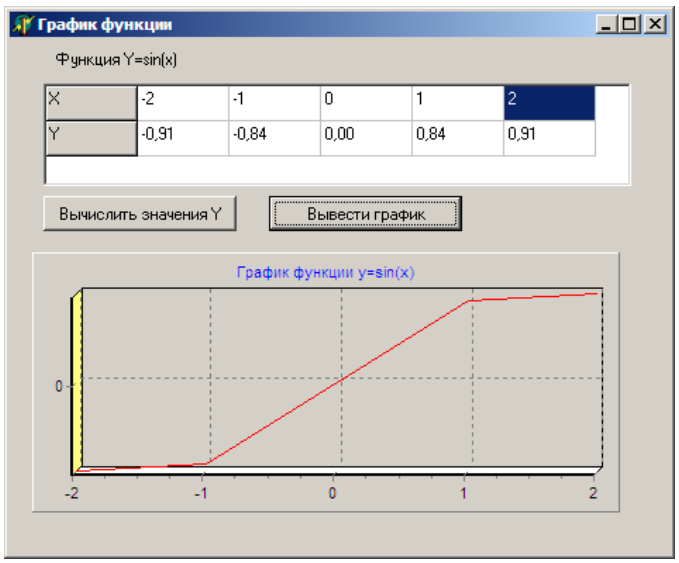


Рисунок 32 – Внешний интерфейс программы

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;
```

6.5.3 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.

6.5.3.1 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.

6.5.3.2 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.

6.5.3.3 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.

6.5.3.4 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, определенная, 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;
```

6.6 Индивидуальные задания

1 Выполнить задания ниже:

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

б) 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:

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

б) 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:

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

б) 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:

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

б) 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:

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

б) 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:

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

б) 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:

а) Построить график функции $y = e^{4x}$, используя компоненты TStringGrid и TChart;

б) 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:

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

б) 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:

а) Построить график функции $y = \cos x + 2$, используя компоненты TStringGrid и TChart;

б) 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:

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

б) 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:

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

б) 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:

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

б) 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:

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

б) 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:

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

б) 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:

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

б) 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»

7 Лабораторная работа №7. Сервер автоматизации MS Word

7.1 Цель работы

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

7.2 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 34, представленного в методических рекомендациях;

4 Для компонента pagecontrol создать две вкладки (п.к.м. по компоненту pagecontrol->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 41;

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.

7.3 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.

7.4 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.

7.5 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.

7.5.1 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
```

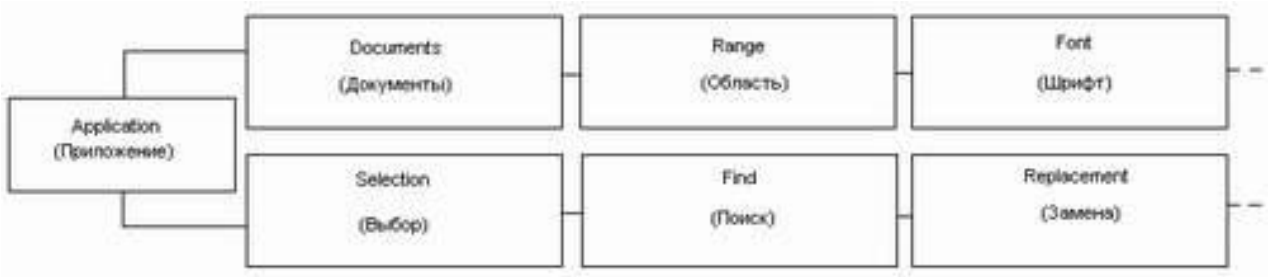


Рисунок 33 – Фрагмент структуры объекта Microsoft Word Object

7.5.2 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.

7.5.2.1 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.

7.5.2.2 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.

7.5.2.3 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 10.

Таблица 10 – 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.

7.5.2.4 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 11.

Таблица 11 – 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.

7.5.2.5 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.

7.5.2.6 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.

7.5.3 Проектирование приложения

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

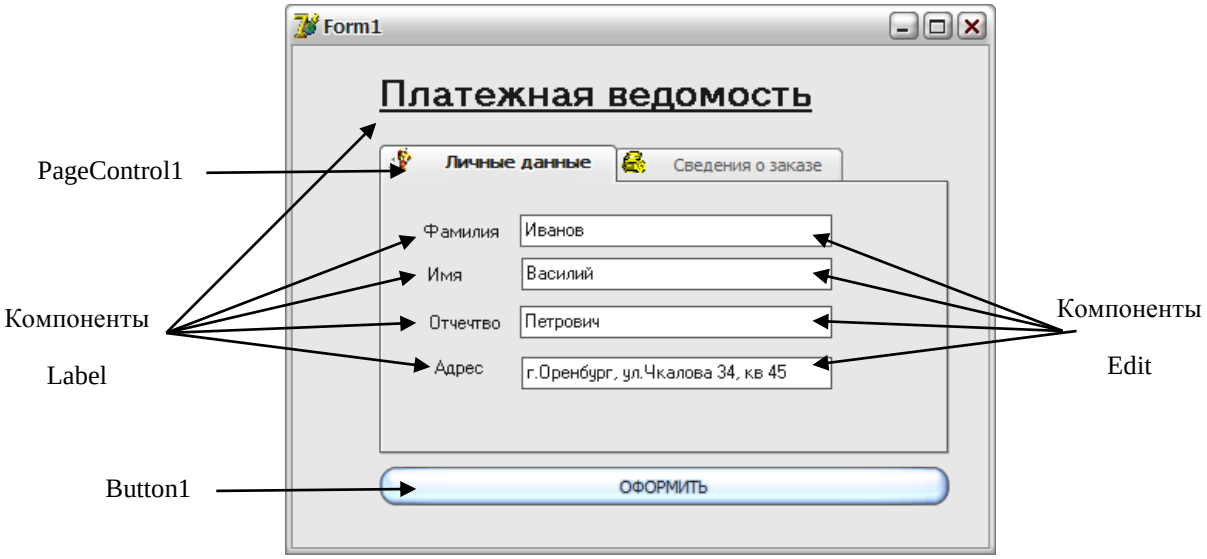


Рисунок 34 – Рекомендуемый внешний вид

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 35).

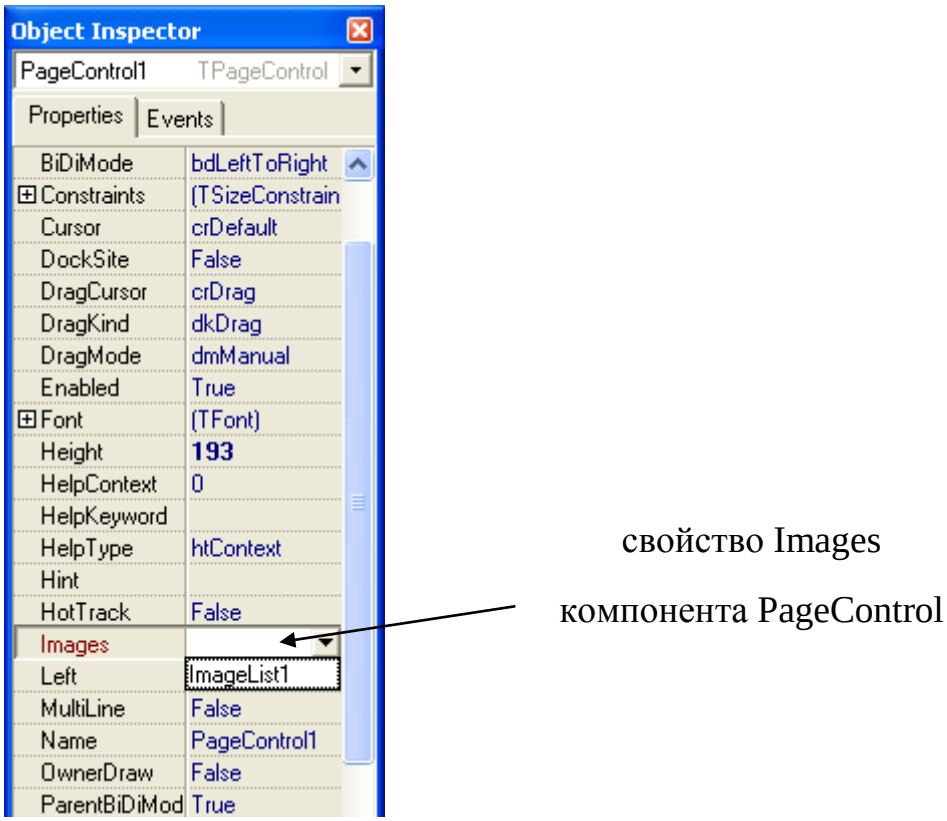


Рисунок 35 – Свойства PageControl

Двойной щелчок по компоненту ImageList1 открывает дополнительное окно, используя которое можно загрузить иконки для вкладок (рисунок 36).

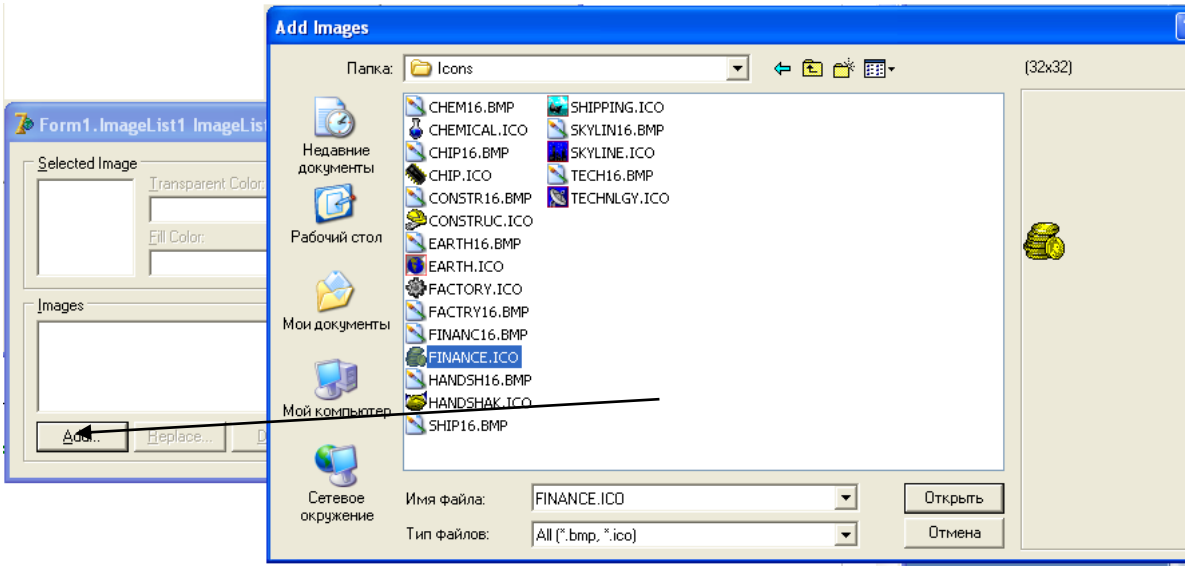


Рисунок 36 – Добавление иконок в ImageList

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

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

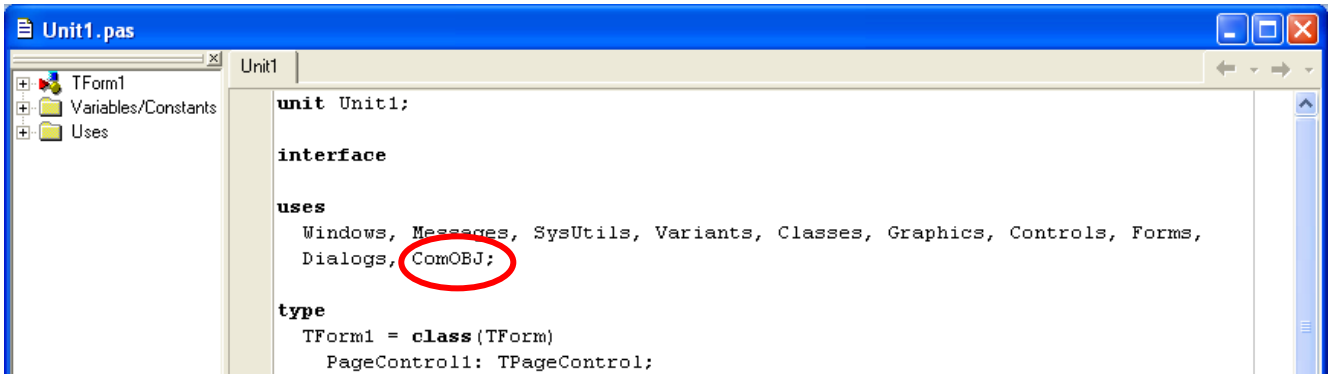


Рисунок 37 – Подключение дополнительного модуля

Создайте шаблон документа Word на основе приведенного ниже:

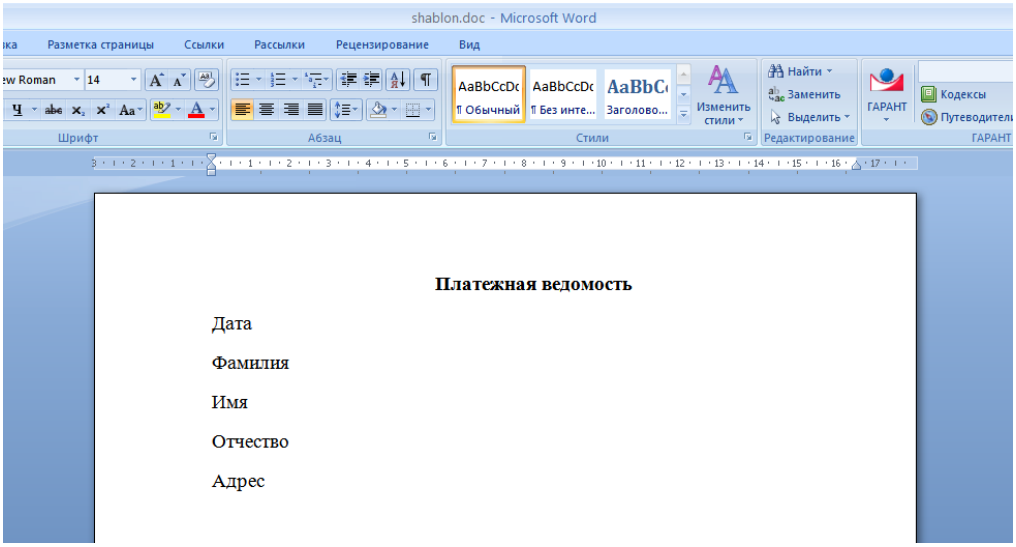


Рисунок 38 – Шаблон документа Word (shablon.doc)

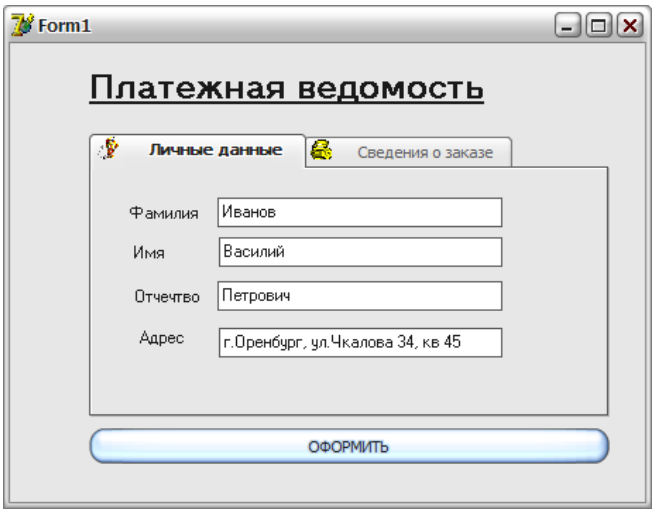


Рисунок 39 – Главная форма приложения

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='Àðåñ' 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.
```

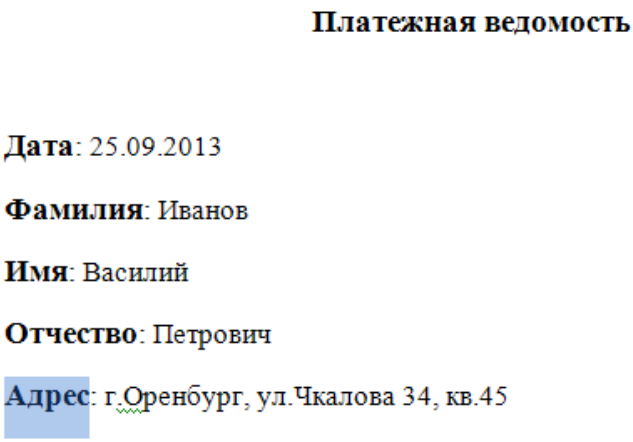


Рисунок 40 – Результат работы проекта

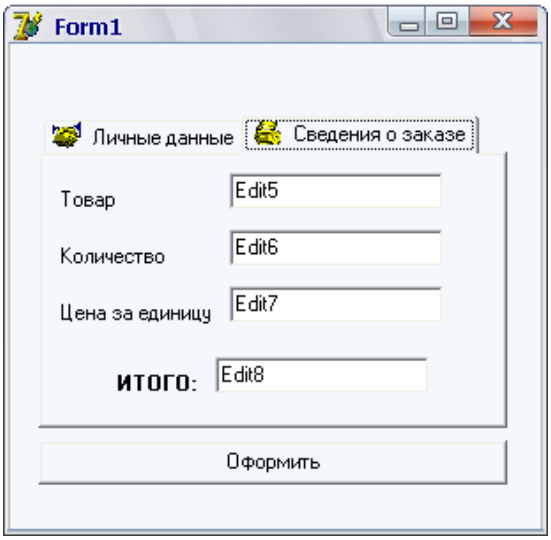


Рисунок 41 – Вкладка «Сведения о заказе»

7.5.4 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);

……

7.6 Индивидуальные задания

1 Создать проект осуществляющий формирование отчета об оборудовании кабинета. Примерный вид отчета приведен на рисунке 42;

Сведения о кабинете

Номер…..

Ответственный _____

ФИО

Наименование кабинета_____

Оборудование

№ п/п	Наименование	Количество	Инвентарный номер

Рисунок 42 – 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.

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

8.1 Цель работы

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

8.2 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 45, 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. 8.5.2*);

7 Организовать фильтрацию записей в таблице по полю *образование* с помощью метода *filter* и *filtered* (методические рекомендации *п. 8.5.3*);

8 Организовать сортировку записей по выбранному столбцу с помощью процедуры *sort* (методические рекомендации *п. 8.5.4*);

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.

8.3 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.

8.4 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?

8.5 Методические рекомендации

8.5.1 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:

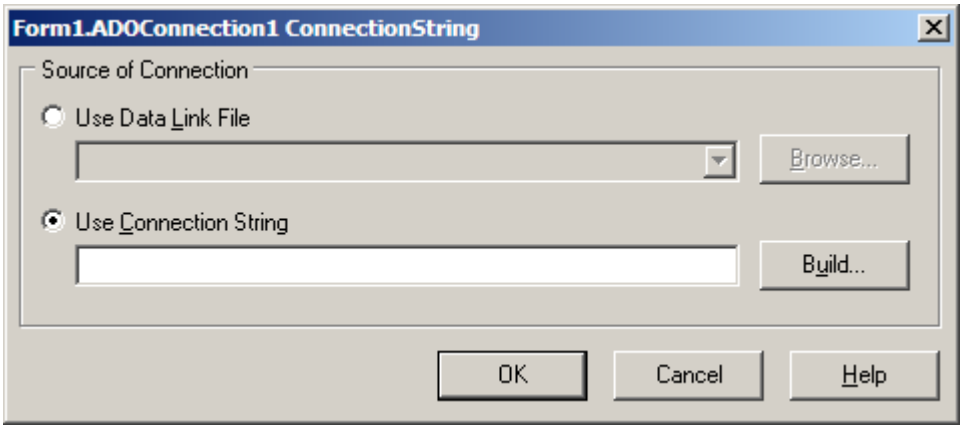


Рисунок 43 – Окно подключения компонента к ADO

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

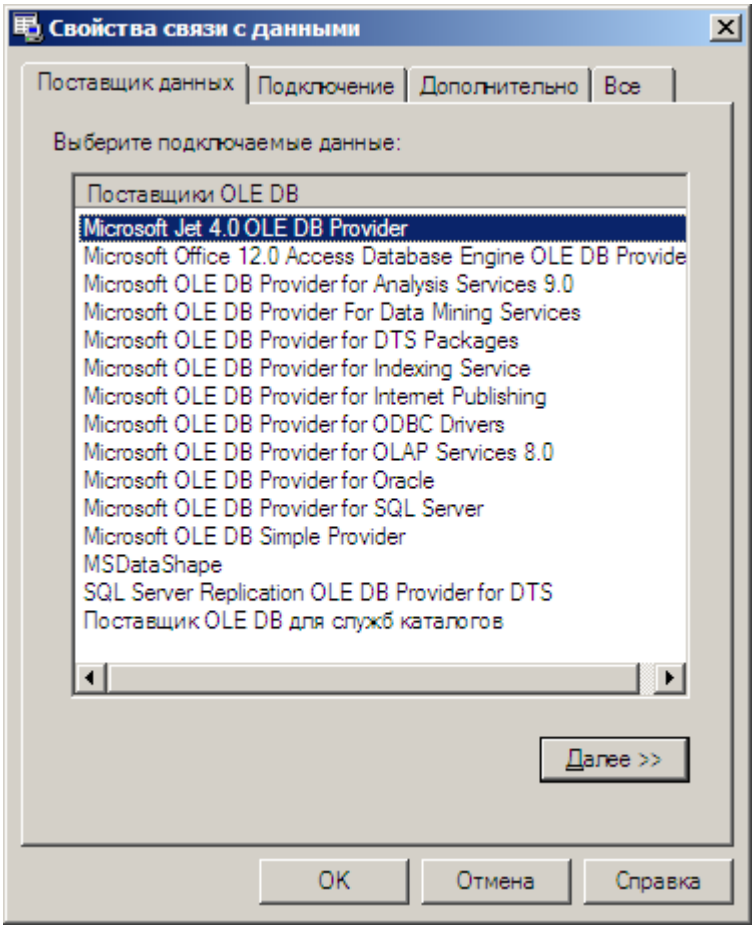


Рисунок 44 – Окно настроек подключения

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 45.

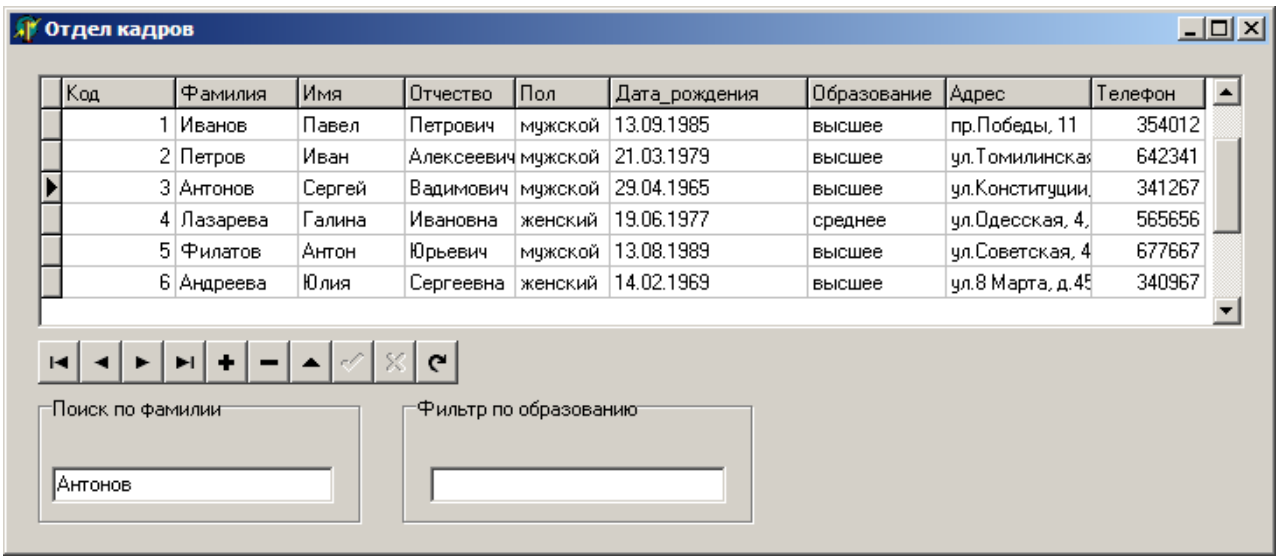


Рисунок 45 – Внешний интерфейс приложения

8.5.2 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.

8.5.3 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;
```

8.5.4 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.

8.6 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 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

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

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

Практикум

Н.А. Уйманова
М.Г. Таспаева

ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

ISBN 978-5-7410-1993-1