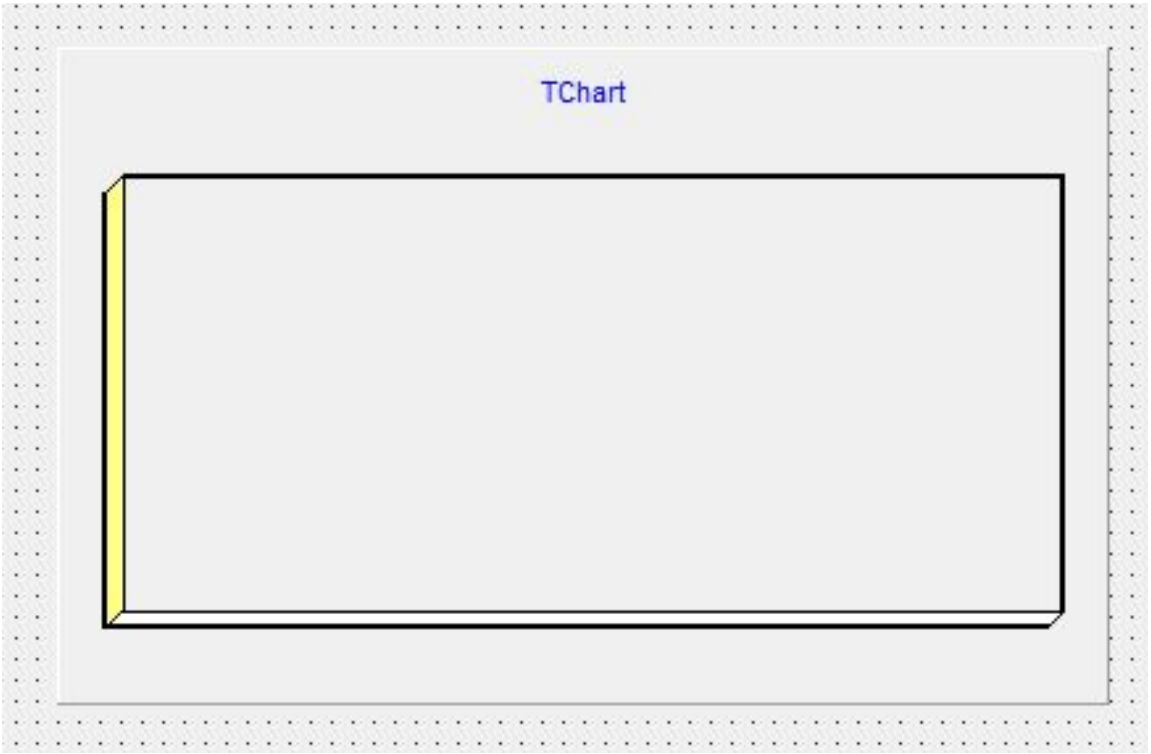
# Система визуального объектно-ориентированного программирования Delphi

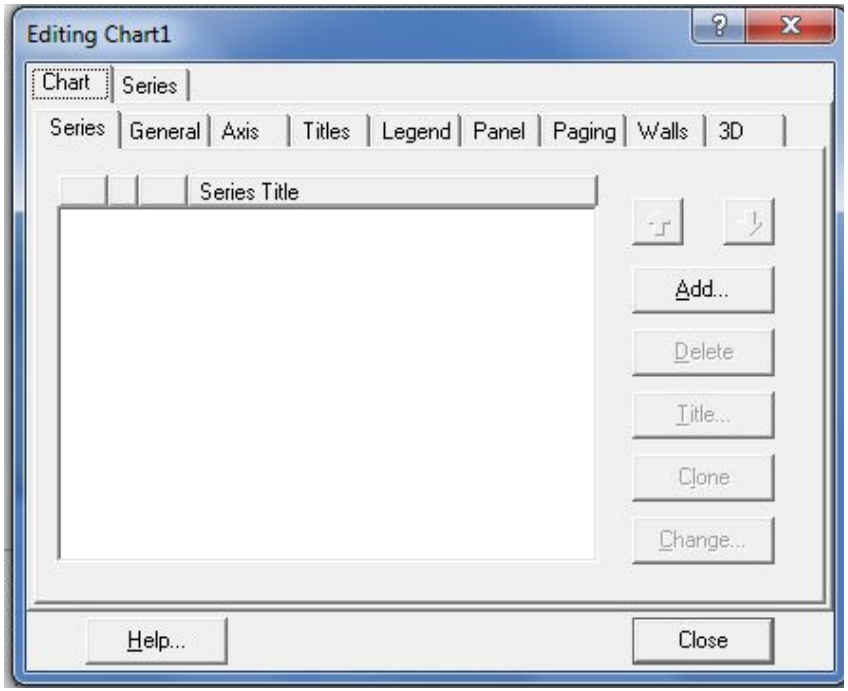
# Компонент Chart

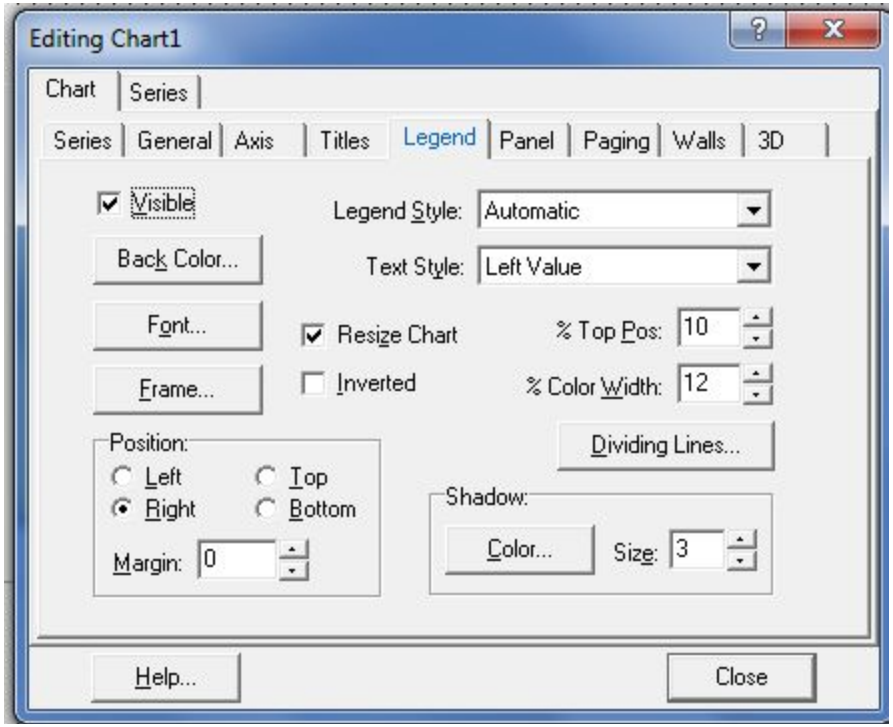
Используется для создания диаграмм и графиков.



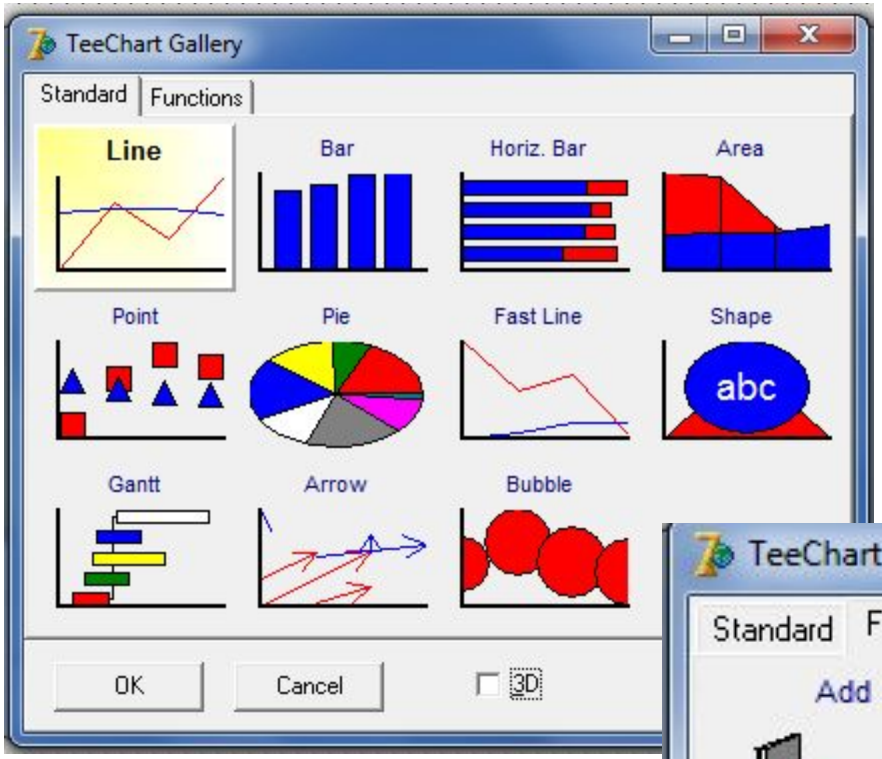
# Свойства Chart

Основное свойство ***SeriesList***, которое устанавливает вид диаграммы и ее настройки

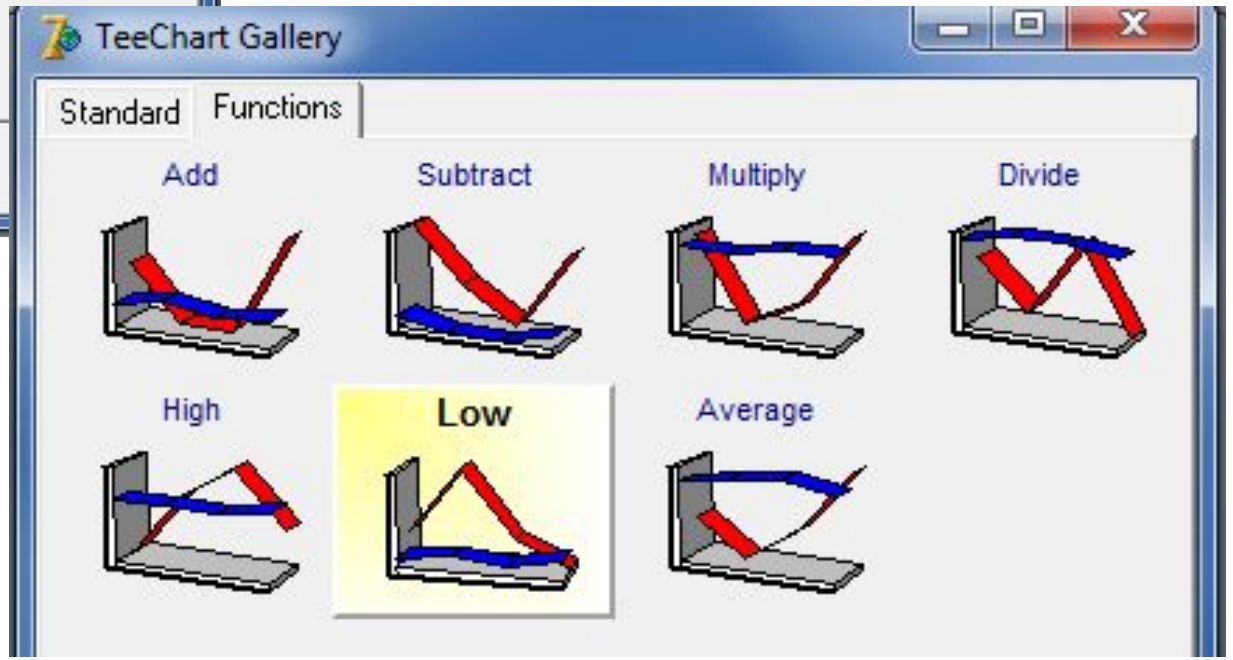



# TreeChart Gallery




Каждая диаграмма это отдельная серия данных, которая может отображаться на экране. Серия может быть одна или несколько



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

- Метод **Clear** очищает серию от занесенных ранее данных.
- Метод **Add**:

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

позволяет добавить в диаграмму новую точку.

Параметр **AValue** соответствует добавляемому значению, параметр **ALabel** — название, которое будет отображаться на диаграмме и в легенде,

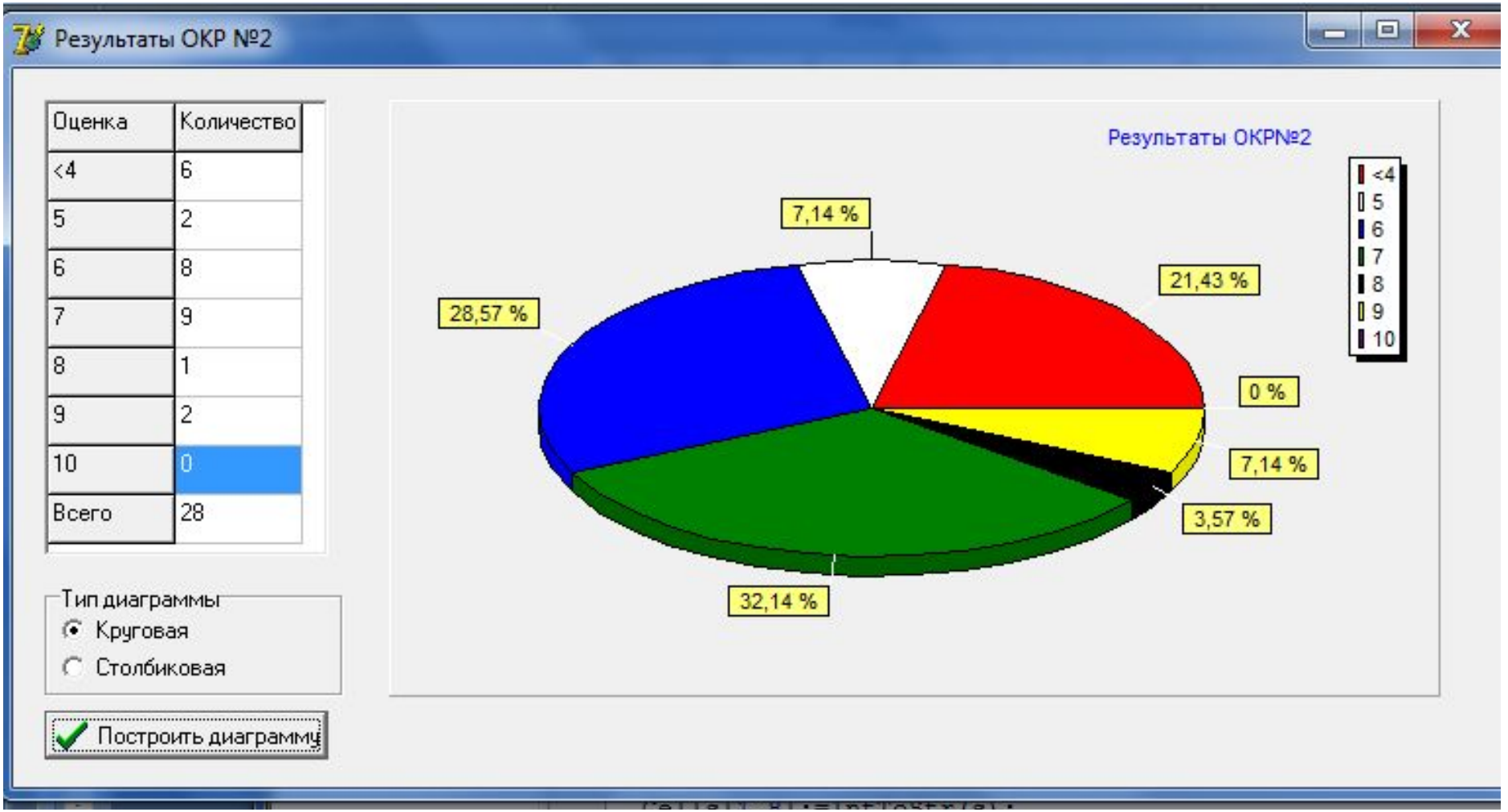
**AColor** — цвет. Параметр **ALabel** — не обязательный, его можно задать пустым: ''.

- Метод **AddXY**:

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

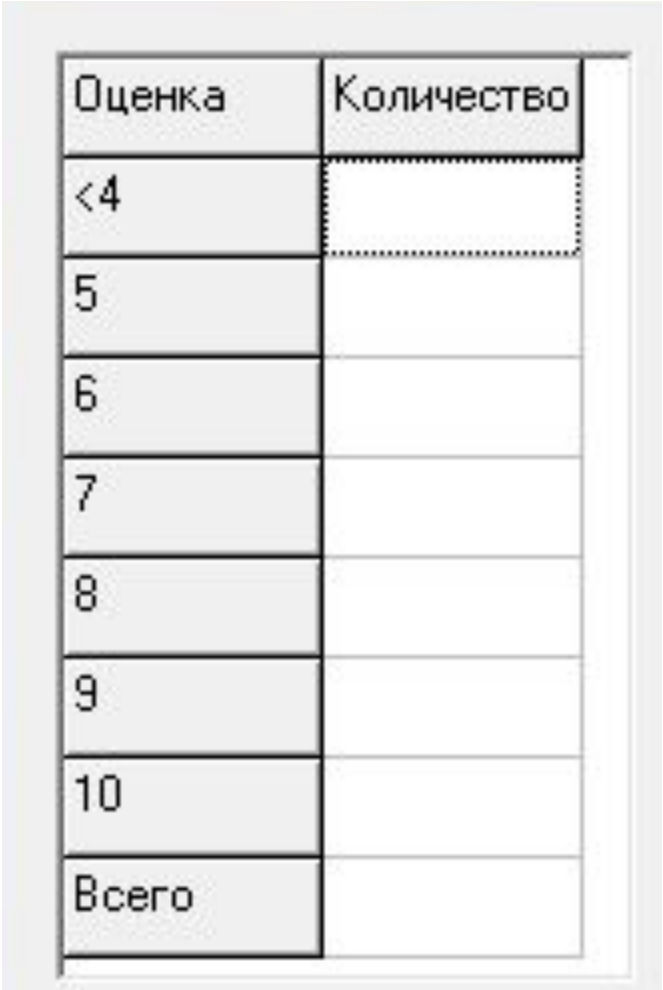
позволяет добавить новую точку в график функции. Параметры **AXValue** и **AYValue** соответствуют аргументу и функции.

# Программа строит диаграммы двух типов по выбору пользователя по результатам контрольной работы



# По созданию формы заполняем таблицу StringGrid

```
procedure TForm1.FormCreate(Sender: TObject);
begin
with StringGrid1 do
begin
Cells[0, 0]:='Оценка';
Cells [1,0]:='Количество';
Cells[0,1]:='<4';
Cells [0,2]:='5';
Cells[0,3]:='6';
Cells[0,4]:='7';
Cells [0,5]:='8';
Cells[0,6]:='9';
Cells [0,7]:='10';
Cells[0,8]:='Всего';
Col:=1;
end;
end;
```



# Обработка события нажатия клавиши: программный подсчет общего числа учащихся

```
var i,s:integer;
    Clr:array [1..7] of TColor=(clRed,clWhite,clBlue,
    clGreen,clBlack,clYellow,clPurple);
begin
//подсчет общего числа учащихся
s:=0;
with StringGrid1 do
begin
  for i:=1 to RowCount-2 do
     inc(s,StrToInt(Cells[1,i]));
  Cells[1,8]:=IntToStr(s);
end;
```

# Обработка события нажатия клавиши: заполнение диаграммы и ее отображение

```
//заполнение диаграммы значениями
with Series1 do
begin
  Clear;
  for i:=1 to StringGrid1.RowCount-2 do
  Add(StrToInt(StringGrid1.Cells[1,i]),StringGrid1.Cells[0,i],Clr[i]);
end;
//заполснение второй серии теми же значениями
Series2.Assign(Series1);
Series2.Active := false;
//выбор радиокнопки
if RadioGroup1.ItemIndex=0
then begin    Series1.Active:=true; Series2.Active:=false;  end
else begin    Series2.Active:=true; Series1.Active:=false;  end;
end;
```

# Системные диалоги (Dialogs)


Страница Dialogs содержит компоненты, используемые для создания различных диалоговых окон, общепринятых в приложениях Windows. Диалоги используются для указания файлов или выбора установок.

# Основные диалоги

| | |
|---|---|
|  **OpenDialog** | Предназначен для создания окна диалога ≪Открыть файл≫ |
|  **SaveDialog** | Предназначен для создания окна диалога ≪Сохранить файл≫. |
|  **OpenPictureDialog** | Предназначен для создания окна диалога ≪Открыть рисунок≫, открывающего графический файл. |
|  **SavePictureDialog** | Предназначен для создания окна диалога ≪Сохранить рисунок≫ - сохранение изображения в графическом файле. |
|  **FontDialog** | Предназначен для создания окна диалога ≪Шрифты≫ — выбор атрибутов шрифта. |
|  **ColorDialog** | Предназначен для создания окна диалога ≪Цвет≫ — выбор цвета. |

# Основные диалоги

| | |
|---|---|
| **ColorBox (Additional)** | Выпадающий список для выбора пользователем цвета. |
| **PrintDialog** | Предназначен для создания окна диалога ≪Печать≫ |
| **PrinterSetupDialog** | Предназначен для создания окна диалога ≪Установка принтера≫. |
| **PageSetupDialog** | Предназначен для создания окна диалога ≪Параметры страницы≫. |
| **FindDialog** | Предназначен для создания окна диалога ≪Найти≫ — контекстный поиск в тексте. |
| **ReplaceDialog** | Предназначен для создания окна диалога ≪Заменить≫ — контекстная замена фрагментов текста. |

# Основные свойства

- ***Execute*** - основной метод, которым производится обращение к любому диалогу.

Эта функция открывает диалоговое окно и, если пользователь произвел в нем какой-то выбор, то функция возвращает true.

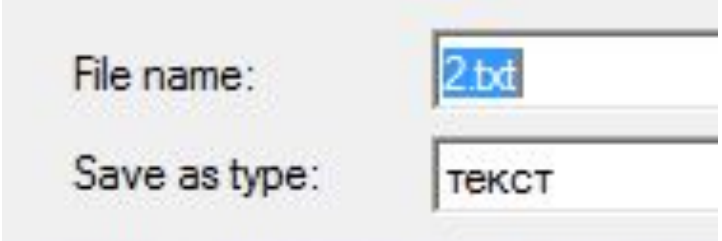
Если же пользователь в диалоге нажал кнопку Отмена или клавишу Esc, то функция Execute возвращает false.

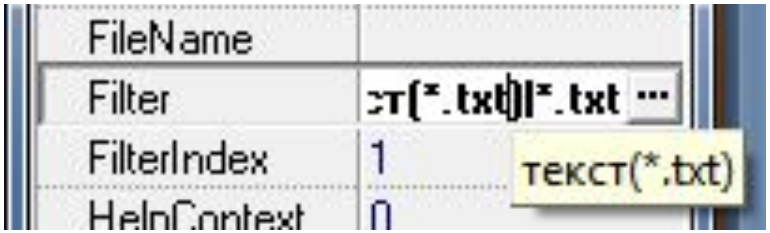
Поэтому стандартное обращение к диалогу имеет вид:

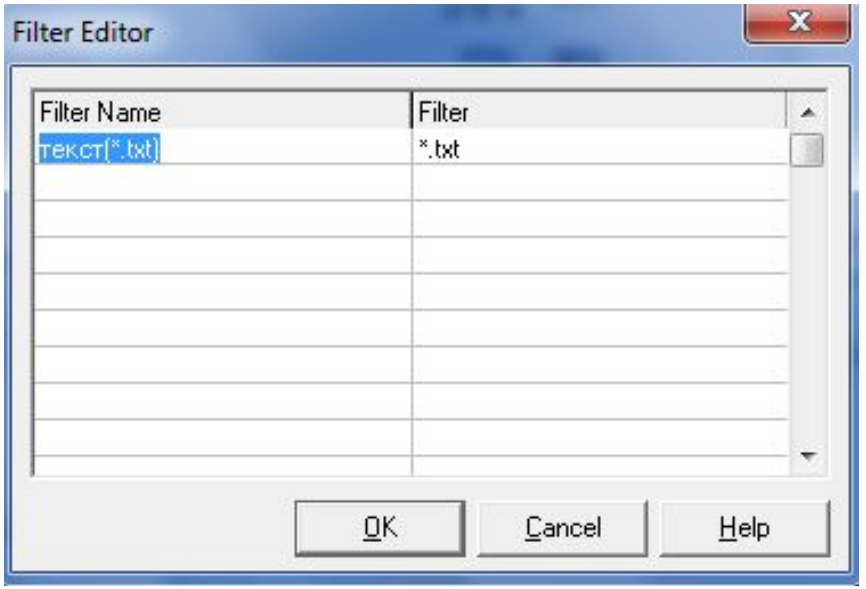
**if <имя компонента-диалога>.Execute**
**then <операторы, использующие выбор пользователя>;**

# Основные свойства

- ***FileName*** – имя выбранного пользователем файла в виде строки.
- ***Filter*** - типы искомых файлов, появляющиеся в диалоге в выпадающем списке <Тип файла>.

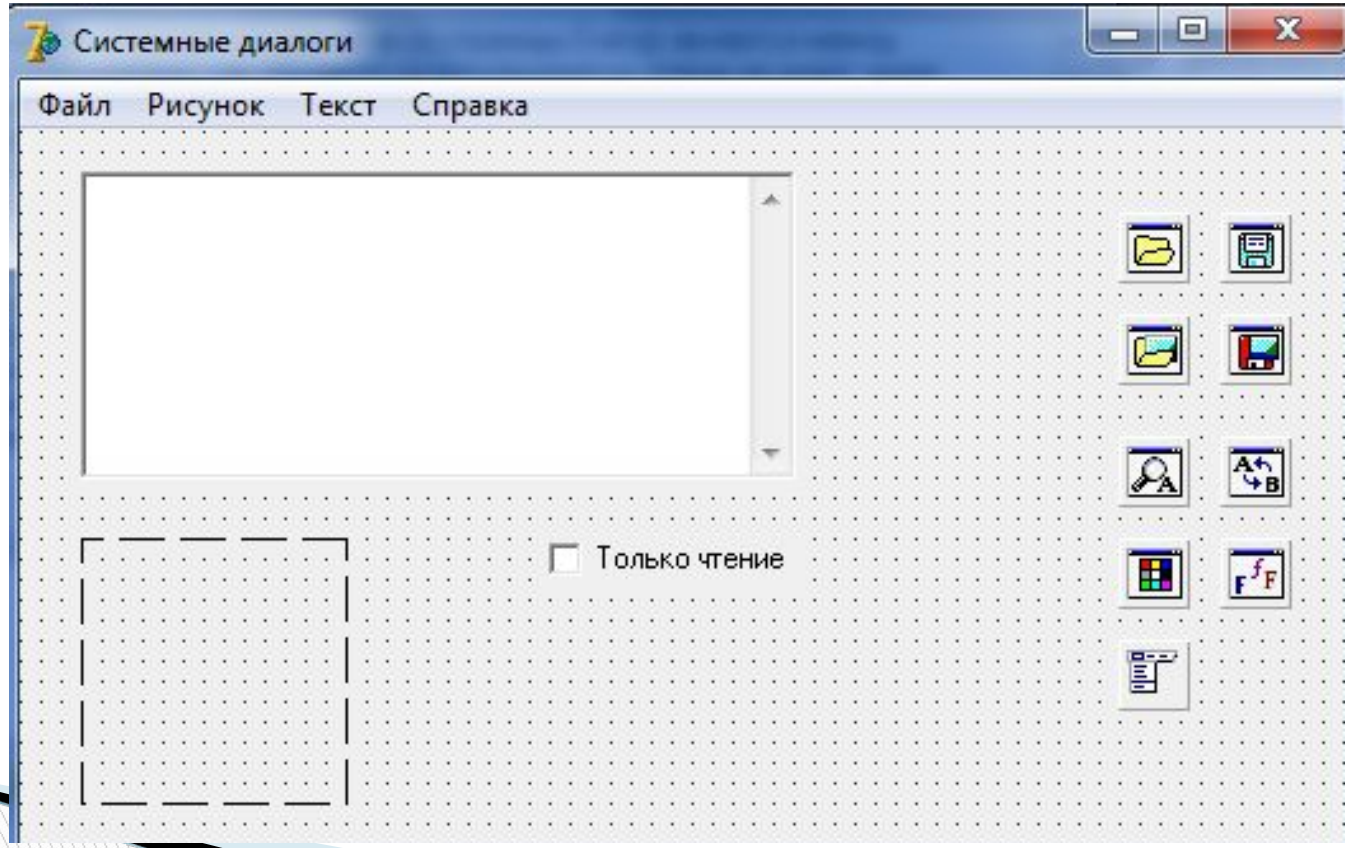


# Основные свойства

Все диалоги являются невизуальными компонентами, и могут быть расположены в любой части окна или скрыты. Пользователю они не видны.



# Диалоги открытия и сохранения файлов — компоненты OpenDialog, SaveDialog, OpenPictureDialog, SavePictureDialog

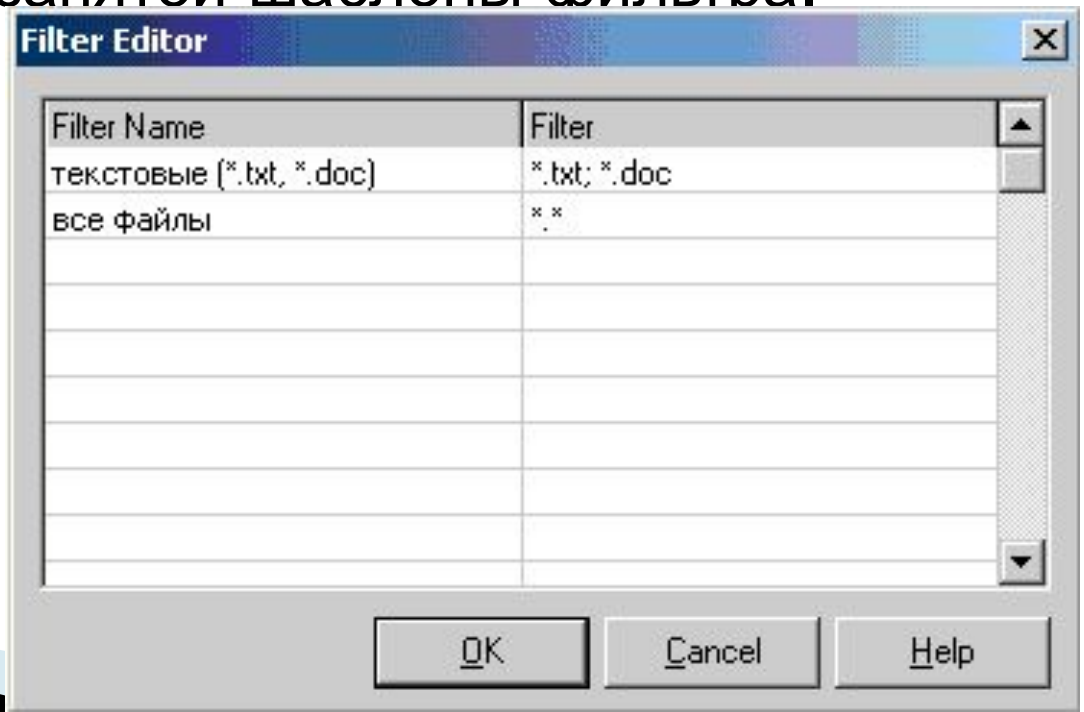
Все свойства этих компонентов одинаковы, только их смысл несколько различен для открытия и закрытия файлов.

Основное свойство, в котором возвращается в виде строки выбранный пользователем файл, — **FileName**. Значение этого свойства можно задать и перед обращением к диалогу. Тогда оно появится в диалоге как значение по умолчанию в окне Имя файла.

Типы искомых файлов, появляющиеся в диалоге в выпадающем списке Тип файла, задаются свойством **Filter**.

В процессе проектирования это свойство проще всего задать с помощью редактора фильтров, который вызывается нажатием кнопки с многоточием около имени этого свойства в Инспекторе Объектов.

При этом открывается окно редактора, в левой панели которого (Filter Name) вы записываете тот текст, который увидит пользователь в выпадающем списке Тип файла диалога. А в правой панели Filter записываются разделенные точками с запятой шаблоны фильтра.



Свойство **FilterIndex** определяет номер фильтра, который будет по умолчанию показан пользователю в момент открытия диалога.

Свойство **InitialDir** определяет начальный каталог, который будет открыт в момент начала работы пользователя с диалогом. Если значение этого свойства не задано, то открывается текущий каталог или тот, который был открыт при последнем обращении пользователя к соответствующему диалогу в процессе выполнения данного приложения.

Свойство **DefaultExt** определяет значение расширения файла по умолчанию. Если значение этого свойства не задано, пользователь должен указать в диалоге полное имя файла с расширением.

Свойство **Title** позволяет вам задать заголовок диалогового окна. Если это свойство не задано, окно открывается с заголовком, определенным в системе.

Свойство **Options** определяет условия выбора файла.

В компонентах диалогов открытия и сохранения файлов предусмотрена возможность обработки ряда событий.

Событие **OnCanClose** возникает при нормальном закрытии пользователем диалогового окна после выбора файла.

При отказе пользователя от диалога — нажатии кнопки Отмена, клавиши Esc и т.д. событие **OnCanClose** не наступает.

В обработке события **OnCanClose** можно произвести дополнительные проверки выбранного пользователем файла и, если по условиям вашей задачи этот выбор недопустим, вы можете известить об этом пользователя и задать значение **false** передаваемому в обработчик параметру **CanClose**. Это не позволит пользователю закрыть диалоговое окно.

Можно также написать обработчики событий **OnFolderChange** — изменение каталога, **OnSelectionChange** — изменение имени файла, **OnTypeChange** — изменение типа файла. В этих обработчиках вы можете предусмотреть какие-то сообщения пользователю.

В Delphi имеются специализированные диалоги открытия и закрытия графических файлов: **OpenPictureDialog** и **SavePictureDialog**.

Диалоговые окна, открываемые этими файлами, отличаются от открываемых компонентами **OpenDialog** и **SaveDialog** удобной возможностью просматривать изображения в процессе выбора файла.

Свойства компонентов **OpenPictureDialog** и **SavePictureDialog** ничем не отличаются от свойств компонентов **OpenDialog** и **SaveDialog**.

Единственное отличие — заданное значение по умолчанию свойства **Filter** в **OpenPictureDialog** и **SavePictureDialog**.

В этих компонентах заданы следующие фильтры:

All (*.jpg; *.jpeg; *.bmp; *.ico; *.emf; *.wmf)

JPEG Image File (*.jpg)

JPEG Image File (*.jpeg)

Bitmaps (*.bmp)

Icons (*.ico)

Enhanced Metafiles (*.emf)

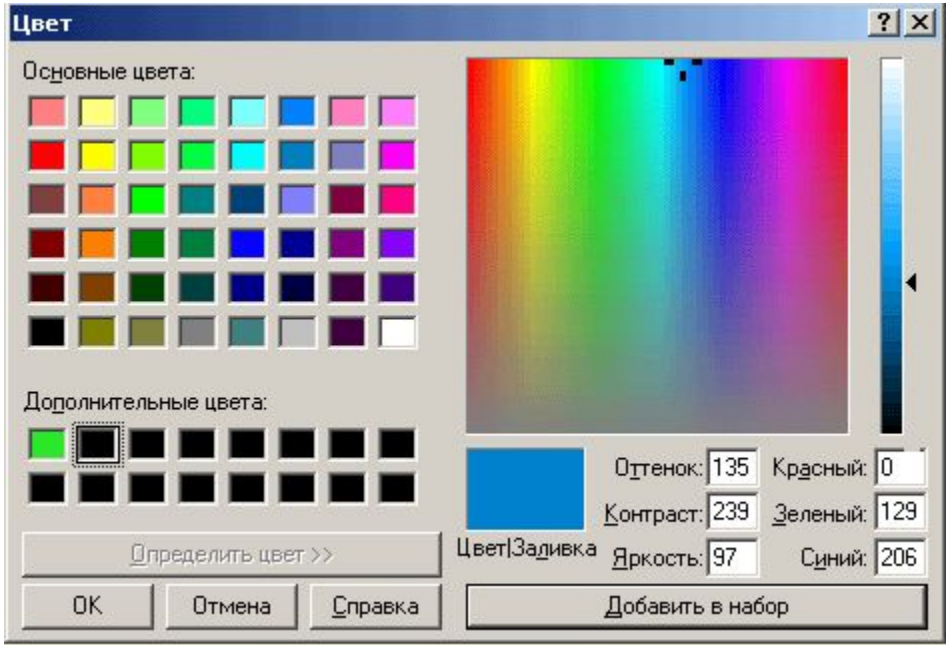
Metafiles (*.wmf).

# Диалог выбора цвета — компонент ColorDialog

Компонент **ColorDialog** вызывает диалоговое окно выбора цвета.

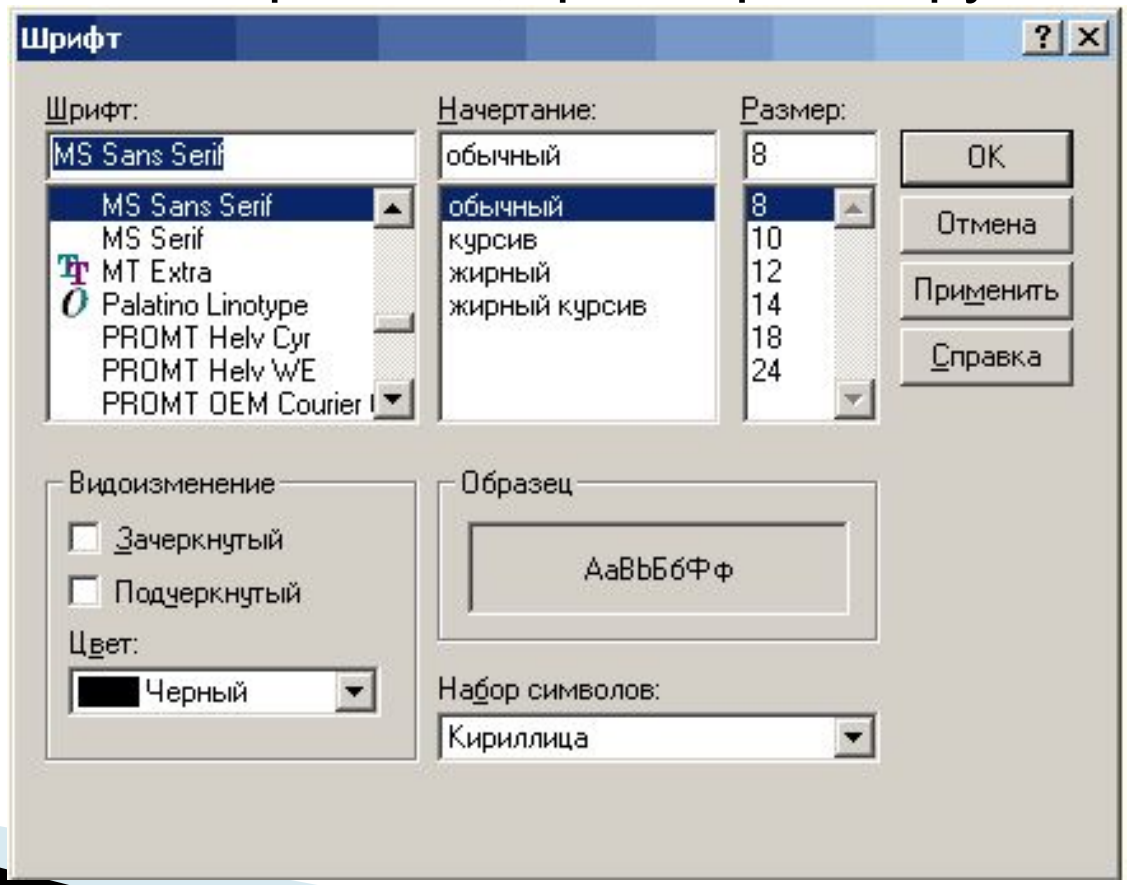
Основное свойство компонента **ColorDialog — Color**.

Это свойство соответствует тому цвету, который выбрал в диалоге пользователь.



# Диалог выбора шрифта — компонент FontDialog

Компонент **FontDialog** вызывает диалоговое окно выбора атрибутов шрифта. В нем пользователь может выбрать имя шрифта, его стиль (начертание), размер и другие атрибуты.



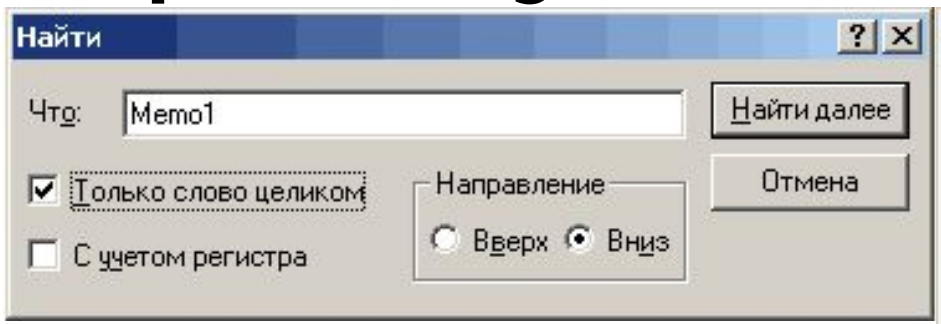
Свойства **MaxFontSize** и **MinFontSize** устанавливают ограничения на максимальный и минимальный размеры шрифта. Если значения этих свойств равны 0 (по умолчанию), то никакие ограничения на размер не накладываются.

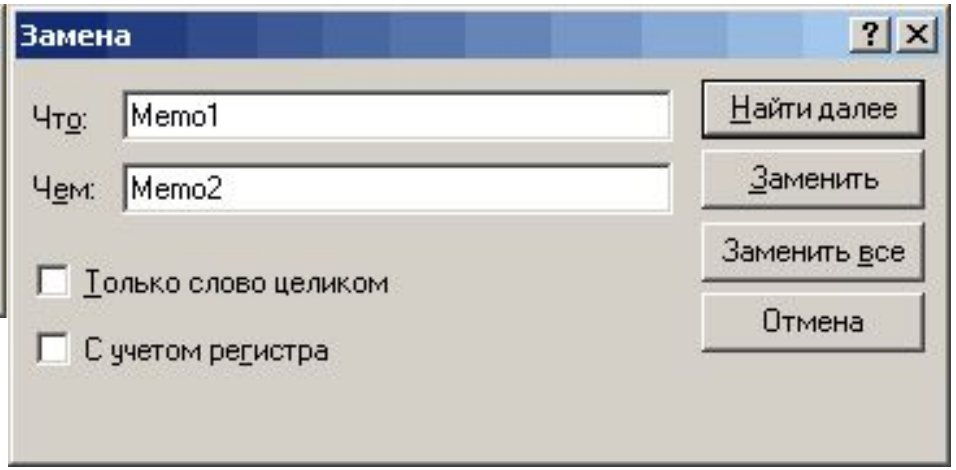
Если же значения свойств заданы, то в списке Размер диалогового окна появляются только размеры, укладывающиеся в заданный диапазон.

При попытке пользователя задать недопустимый размер ему будет выдано предупреждение вида «Размер должен лежать в интервале ...» и выбор пользователя отменится.

# Диалоги поиска и замены текста — компоненты FindDialog и ReplaceDialog

Компоненты **FindDialog** и **ReplaceDialog**, вызывающие диалоги поиска и замены фрагментов текста, очень похожи и имеют одинаковые свойства, кроме одного, задающего заменяющий текст в компоненте **ReplaceDialog**.



Компоненты имеют следующие основные свойства:

**FindText** - Текст, заданный пользователем для поиска или замены. Программно может быть установлен как начальное значение, предлагаемое пользователю.

**ReplaceText** - Только в компоненте **ReplaceDialog** — текст, который должен заменять **FindText.**

**Position** - Позиция левого верхнего угла диалогового окна, заданная типом **TPoint** — записью, содержащей поля **X** (экранная координата по горизонтали) и **Y** (экранная координата по вертикали) .

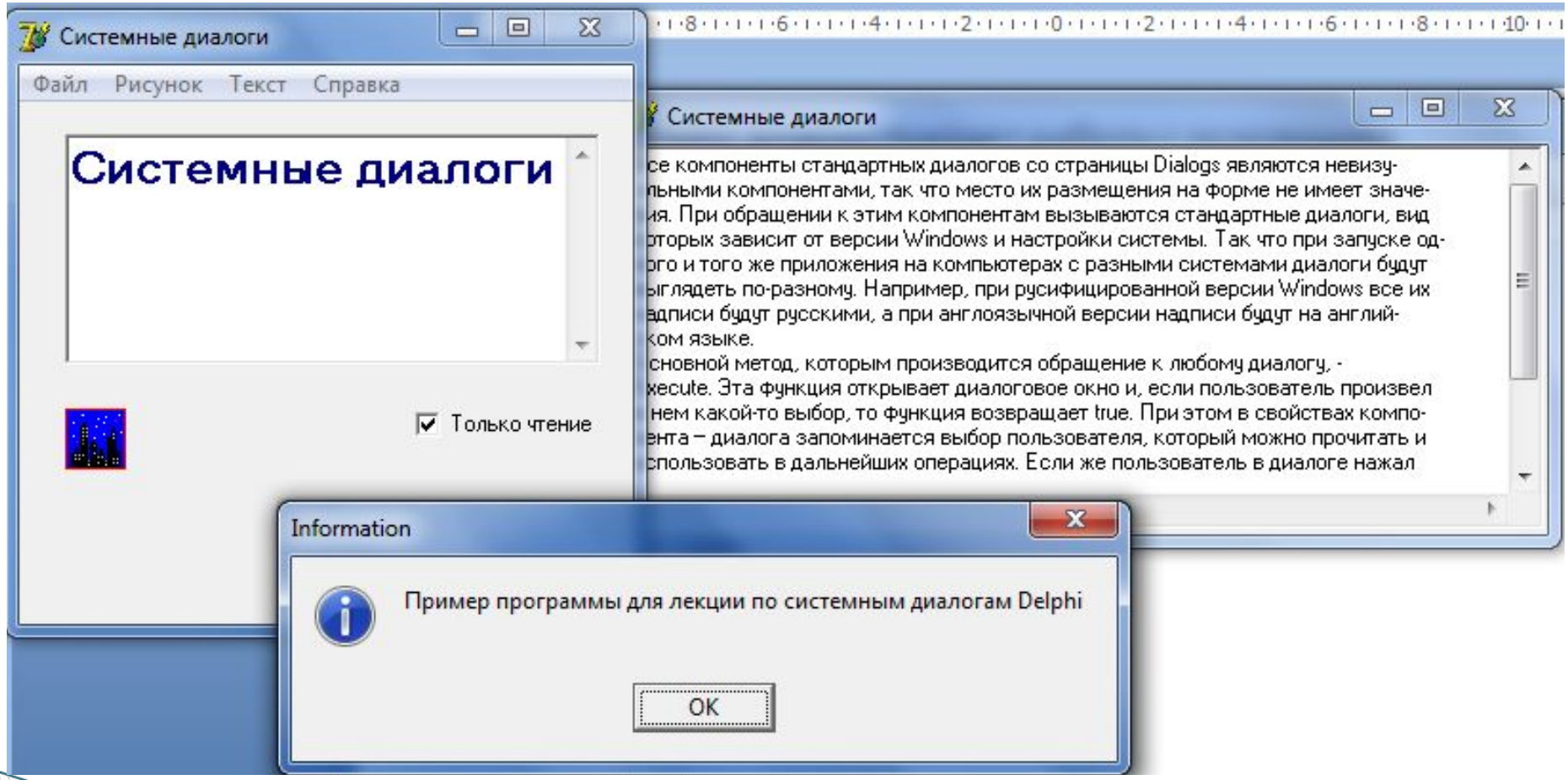
**Options** - Множество опций.

Сами по себе компоненты **FindDialog** и **ReplaceDialog** не осуществляют ни поиска, ни замены. Они только обеспечивают интерфейс с пользователем. А поиск и замену надо осуществлять программно.

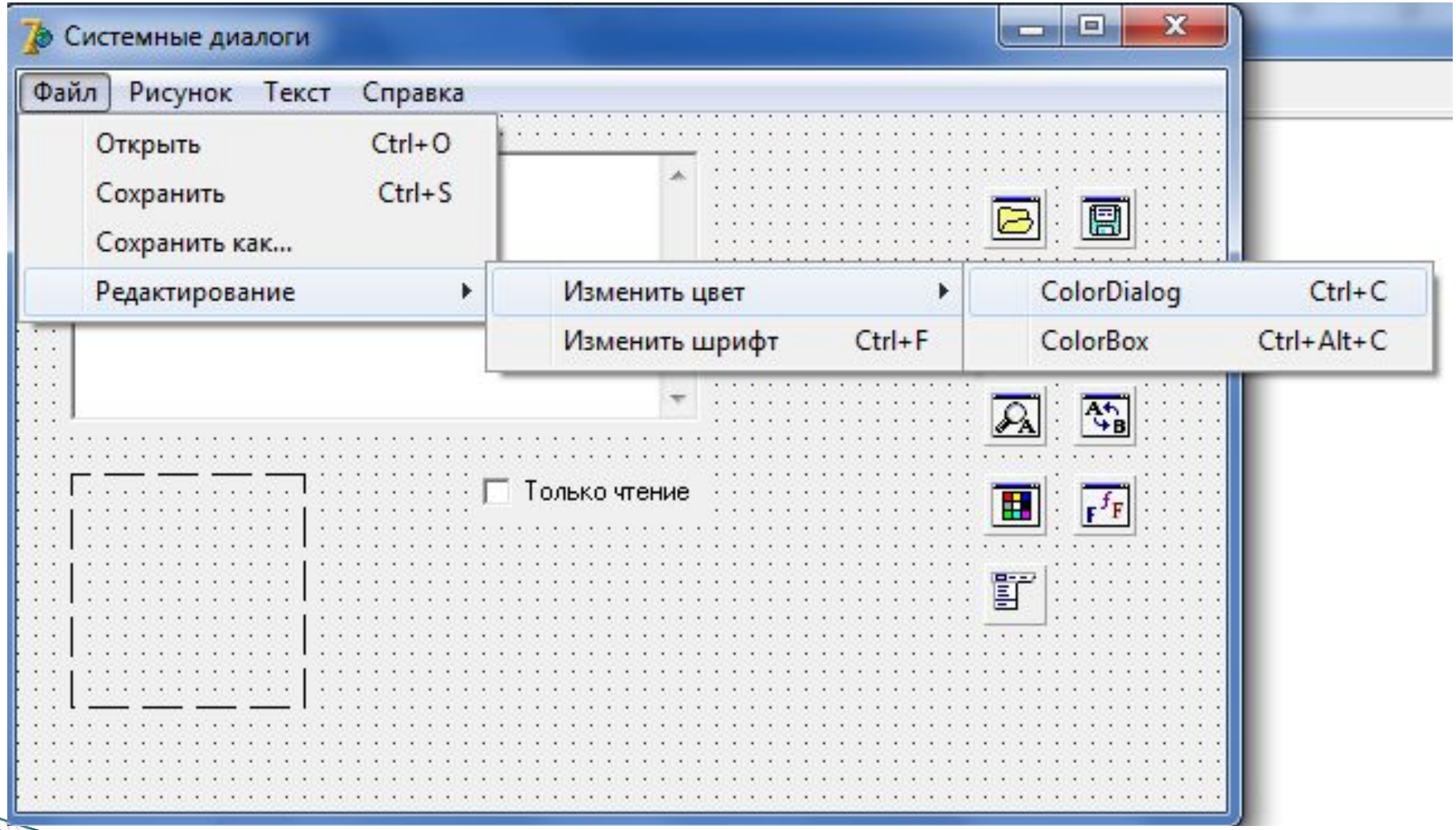
Для этого можно пользоваться событием **OnFind**, происходящим, когда пользователь нажал в диалоге кнопку Найти далее, и событием **OnReplace**, возникающим, если пользователь нажал кнопку Заменить или Заменить все.

В событии **OnReplace** узнать, какую именно кнопку нажал пользователь, можно но значениям флагов **frReplace** и **frReplaceAll**.

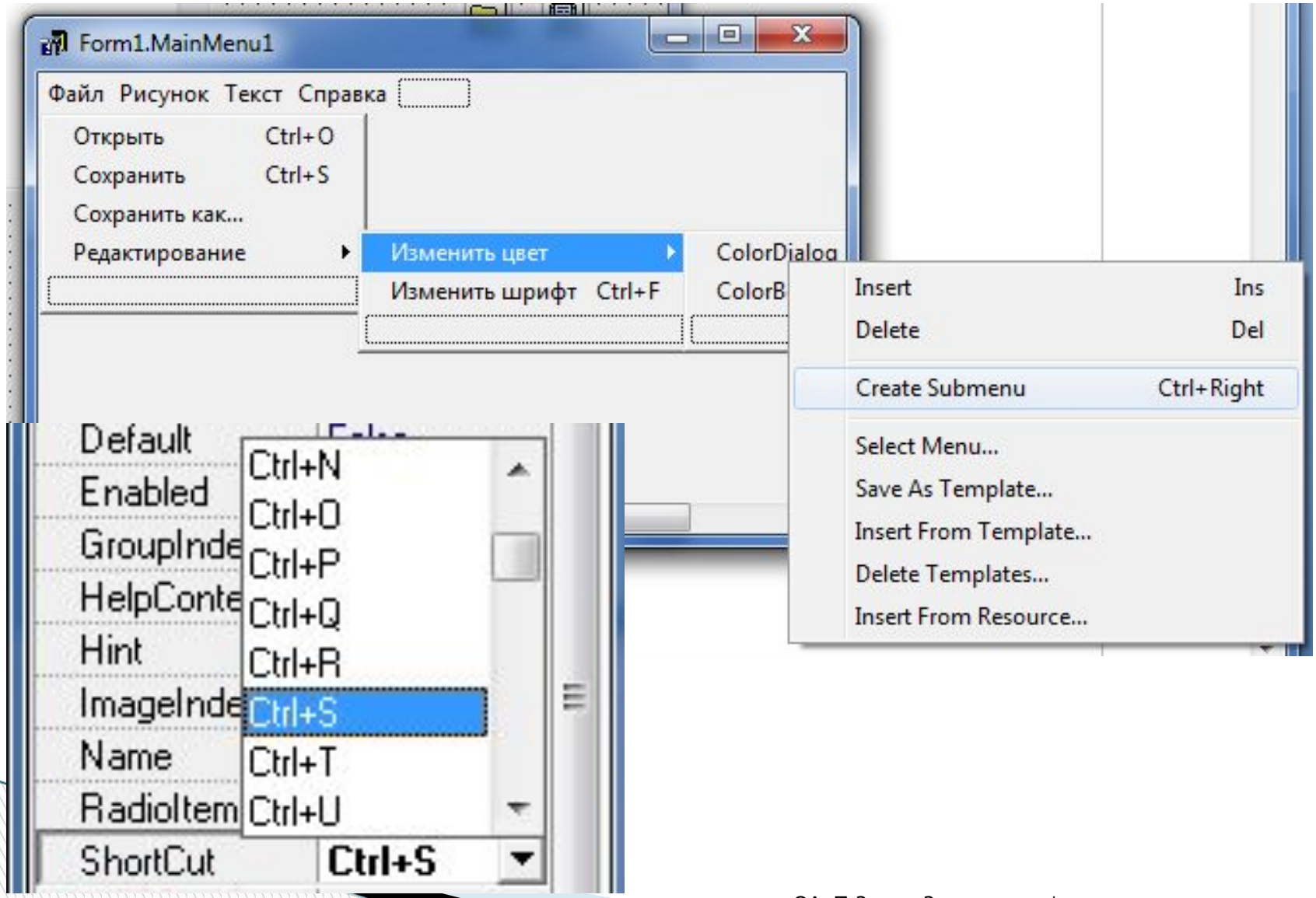
Программа демонстрирует работу с основными диалогами, меню, компонентами Memo, CheckBox, Image, ColorBox и разными формами.



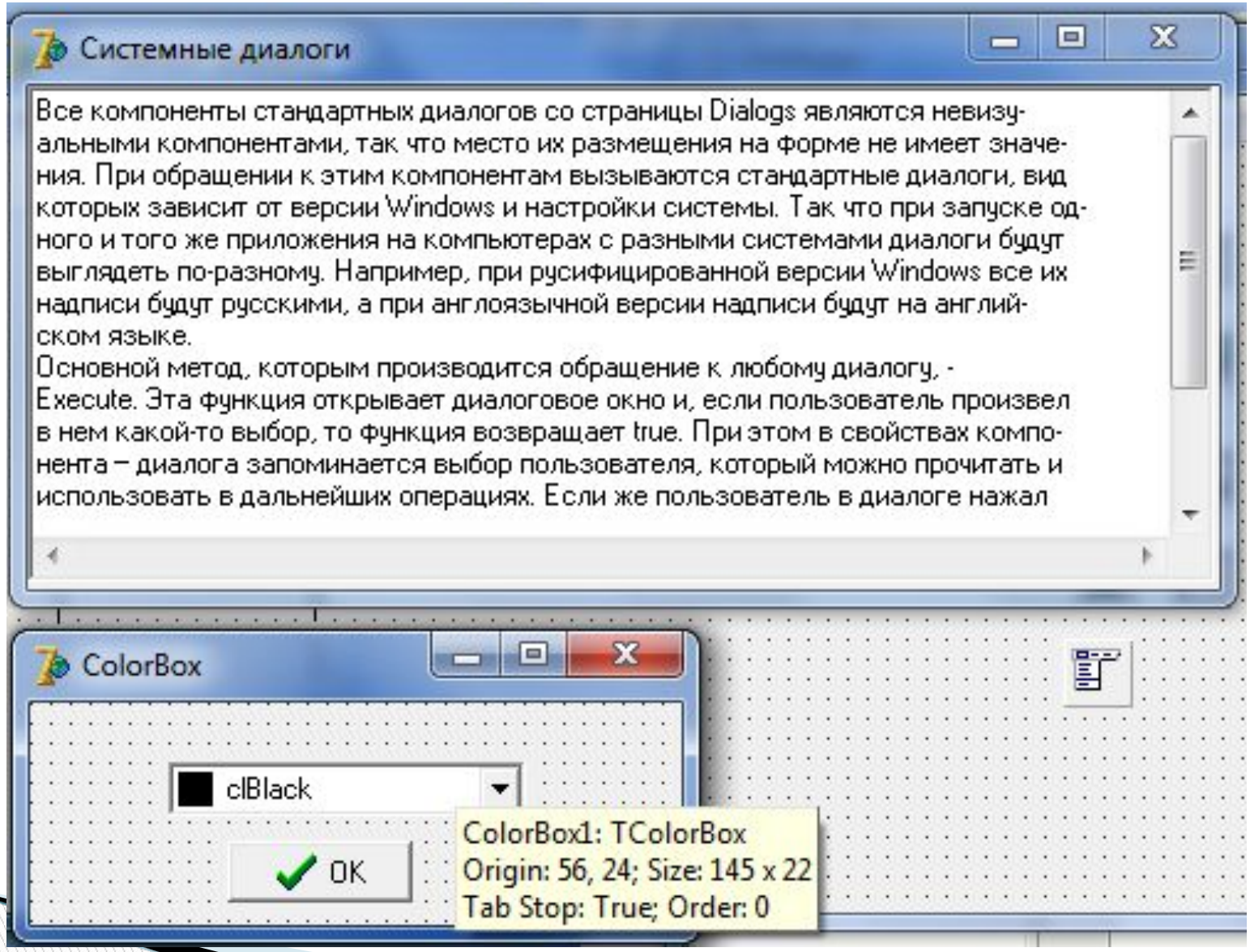
# Размещаем на форме все нужные компоненты и создаем меню



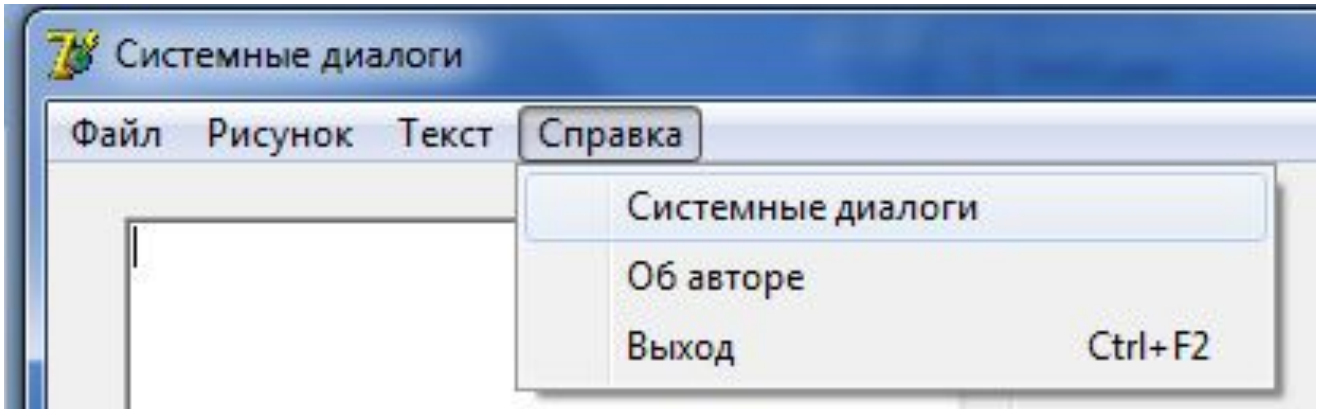
# Создание контекстного меню и быстрых клавиш



# Создание дополнительных форм – Form2, Form3



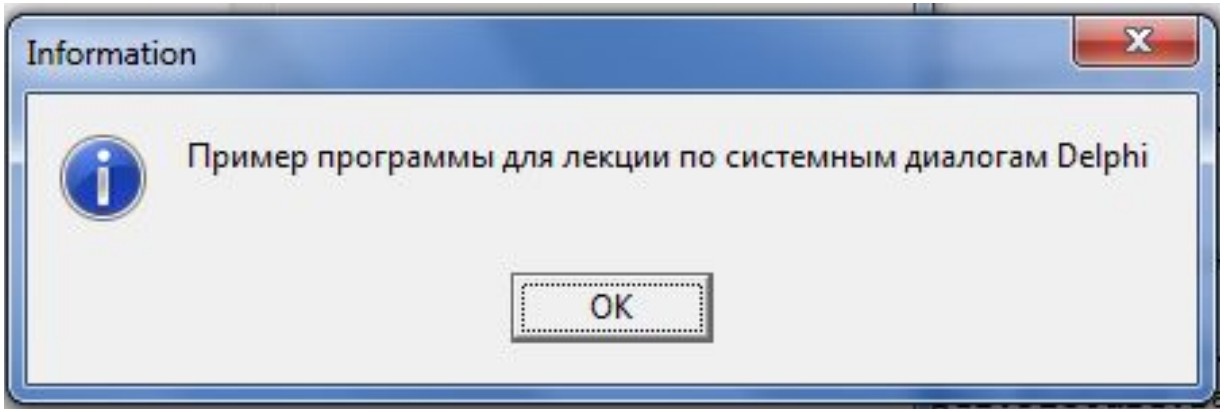
# Обработка меню
# Справка | Системные диалоги




```
procedure TForm1.N11Click(Sender: TObject);
begin
Form2.Show;
end;
```

# Обработка меню
## Справка | Об авторе

```
procedure TForm1.N3Click(Sender: TObject);
begin
  MessageDlg('Пример программы для лекции по
  системным диалогам Delphi',mtInformation,[mbOk],0);
end;
```



Функция MessageDlg используется для вывода сообщения пользователю

function MessageDlg ( const Message : string; DialogType : TMsgDlgType; Buttons : TMsgDlgButtons; HelpContext : Longint ) : Integer;

Вызов MessageDlg выводит на экран диалоговое окно и ожидает ответа пользователя. Сообщение в окне задается параметром функции Message

Вид отображаемого окна задается параметром DialogType. Возможные значения этого параметра:

mtWarning - Окно замечаний (желтый восклицательный знак)

mtError - Окно ошибок (красный стоп-сигнал).

mtInformation - Информационное окно (голубой символ “I”)

mtConfirmation - Окно подтверждения (зеленый вопросительный знак)

mtCustom - Заказное окно без рисунка. Заголовок соответствует имени выполняемого файла приложения.

Параметр Buttons определяет, какие кнопки будут присутствовать в окне. Тип TMsgDlgBtns параметра AButtons является множеством, которое включает различные кнопки. Возможные значения видов кнопок:

| | |
|---|---|
| mbYes | Кнопка с надписью 'Yes' |
| mbNo | Кнопка с надписью 'No' |
| mbOK | Кнопка с надписью 'OK' |
| mbCancel | Кнопка с надписью 'Cancel' |
| mbHelp | Кнопка с надписью 'Help' |
| mbAbort | Кнопка с надписью 'Abort' |
| mbRetry | Кнопка с надписью 'Retry' |
| mbIgnore | Кнопка с надписью 'Ignore' |
| mbAll | Кнопка с надписью 'All' |

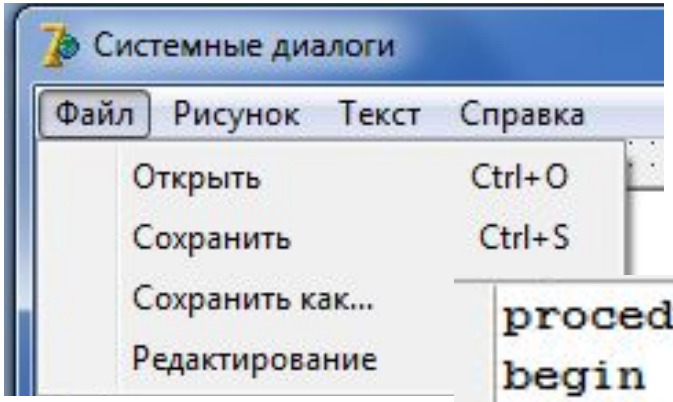
Список необходимых кнопок заключается в квадратные скобки [ ], поскольку параметр Buttons является множеством.

Параметр HelpContext определяет экран контекстной справки, соответствующий данному диалоговому окну. Этот экран справки будет появляться при нажатии пользователем клавиши F1. Если вы справку не планируете, при вызове MessageDlg надо задать нулевое значение параметра HelpContext.

Функция MessageDlg возвращает значение, соответствующее выбранной пользователем кнопке. Возможные возвращаемые значения:

| | | |
|---|---|---|
| mrNone | mrAbort | mrYes |
| mrOk | mrRetry | mrNo |
| mrCancel | mrIgnore | mrAll |

# Обработка меню
# Файл | Открыть и Файл | Сохранить




```
procedure TForm1.N5Click(Sender: TObject);
begin
  if OpenDialog1.Execute
  then
  begin
    fname:=OpenDialog1.FileName;
    Memo1.Lines.LoadFromFile(fname);
  end;
end;
//Сохранить файл
procedure TForm1.C1Click(Sender: TObject);
begin
  Memo1.Lines.SaveToFile(fname);
end;
```

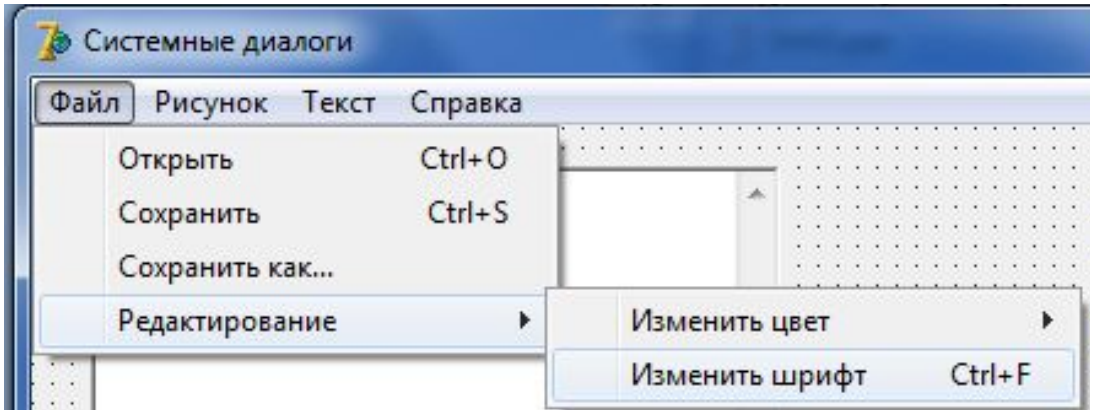
type System.T

# Обработка меню
## Файл | Сохранить как

Unit1 | Unit2 | Unit3

```
procedure TForm1.N16Click(Sender: TObject);
begin
   if SaveDialog1.Execute
   then
   begin
     fname:=SaveDialog1.FileName+'.txt';
     Memo1.Lines.SaveToFile(fname);
   end;
end;
```

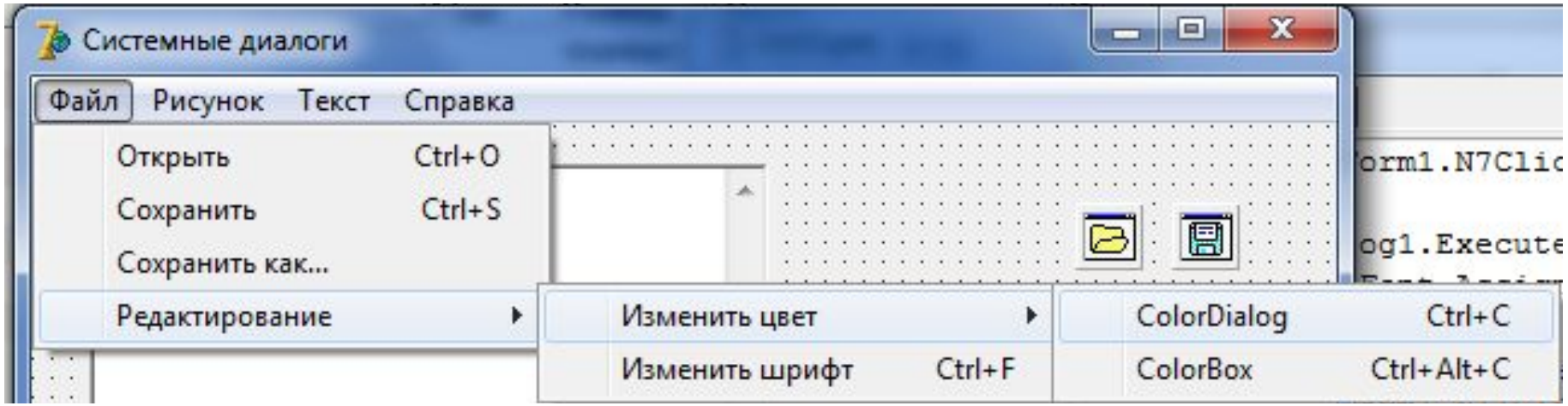
# Обработка меню
## Файл | Редактирование | Изменить шрифт




```
procedure TForm1.N7Click(Sender: TObject);
begin
 if FontDialog1.Execute
  then Memo1.Font.Assign(FontDialog1.Font);
end;
```

# Обработка меню

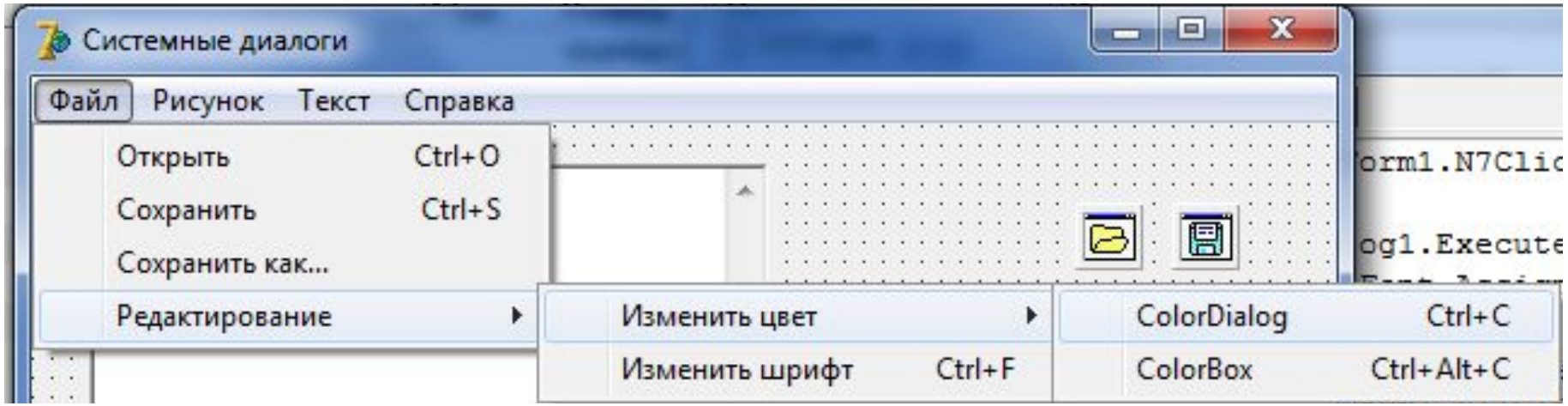
## Файл | Редактирование | Изменить цвет | ColorDialog




```
procedure TForm1.ColorDialog2Click(Sender: TObject);
begin
  if ColorDialog1.Execute
  then Memo1.Font.Color:=ColorDialog1.Color;
end;
```

# Обработка меню

## Файл | Редактирование | Изменить цвет | ColorBox




```
procedure TForm1.ColorBox2Click(Sender: TObject);
begin
  Form3.ShowModal;
end;
```

# Обработка меню

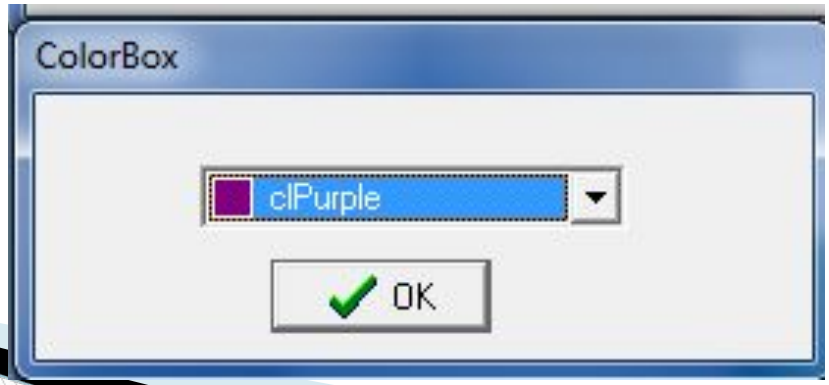
## Файл | Редактирование | Изменить цвет | ColorBox

Unit1 | Unit2 | Unit3

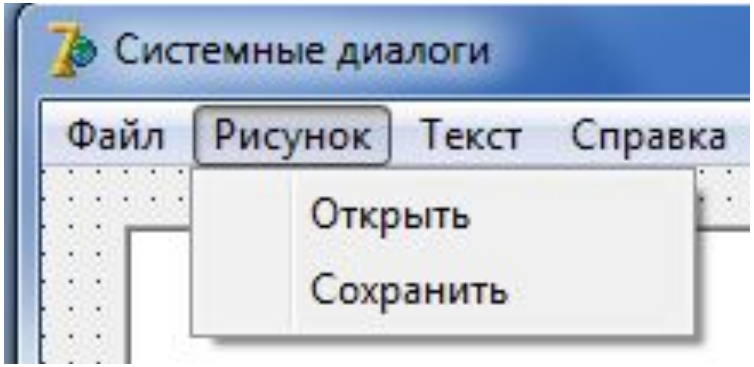
```
implementation

uses Unit1;

{$R *.dfm}
//Изменение цвета с помощью ColorBox
procedure TForm3.BitBtn1Click(Sender: TObject);
begin
  Form1.Memo1.Font.Color:=Form3.ColorBox1.Selected;
end;
```

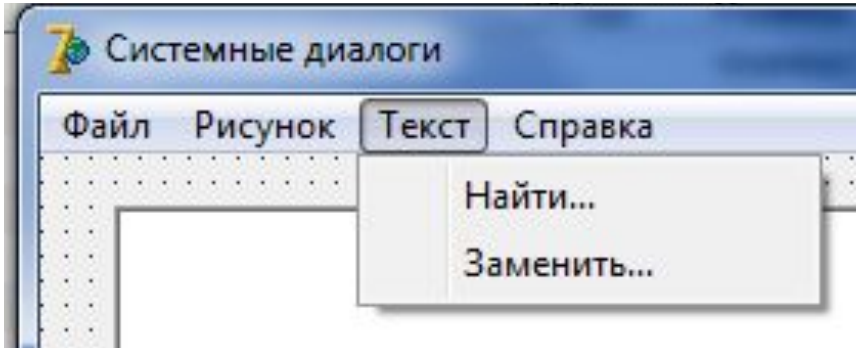


# Обработка меню
# Рисунок | Открыть и Рисунок | Сохранить



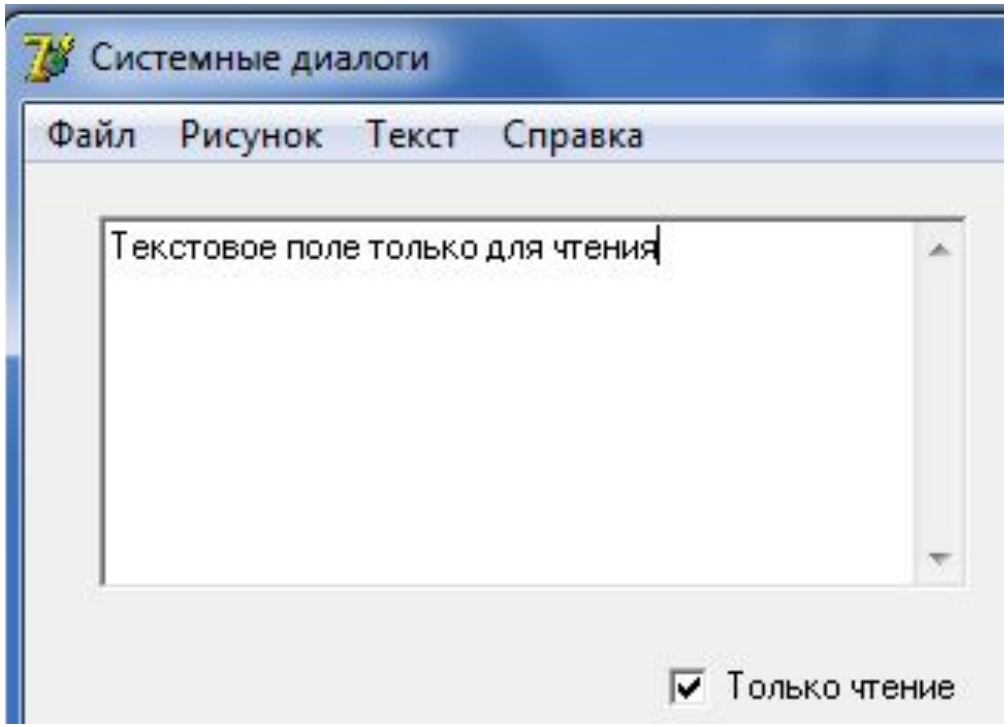

```
//Открыть изображение
procedure TForm1.N9Click(Sender: TObject);
begin
  if OpenPictureDialog1.Execute
     then Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
end;
//Сохранить изображение
procedure TForm1.N10Click(Sender: TObject);
begin
   if SavePictureDialog1.Execute
   then Image1.Picture.SaveToFile(SavePictureDialog1.FileName);
end;
```

# Обработка меню
# Текст | Найти и Текст | Заменить




```
//Запустить диалог поиска
procedure TForm1.N14Click(Sender: TObject);
begin
  FindDialog1.Execute;
end;
//Запустить диалог замены
procedure TForm1.N15Click(Sender: TObject);
begin
  ReplaceDialog1.Execute;
end;
```

# Обработка индикатора Только чтение




```
//Установить Memo1 только для чтения
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  Memo1.ReadOnly:=CheckBox1.Checked;
end;
```

# Timer

Позволяет задавать в приложении интервалы времени. Таймер — невизуальный компонент.

Свойства:

***Interval*** – интервал времени в миллисекундах

***Enabled*** - доступность.

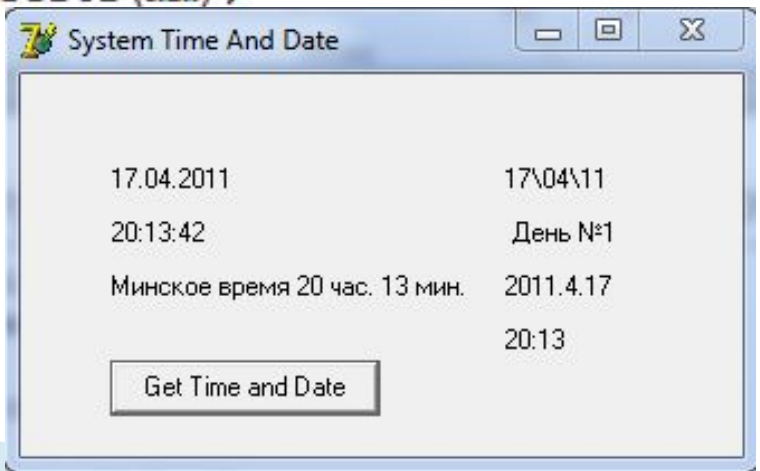
Если Interval = 0 или Enabled=false, то таймер перестает работать.

Основное событие – ***OnTimer***. В нем записываются операторы, которые должны выполняться по запуску таймера (по истечению времени, указанному в Interval).

Чтобы запустить отсчет времени надо или задать **Enabled = true**, если установлено положительное значение **Interval**, или задать положительное значение **Interval**, если **Enabled = true**.

# Способы получения системной даты и времени

```
procedure TForm1.Button1Click(Sender: TObject);
Var DT:TDateTime;
    y,m,d:word;
    hh,mm,ss,ms:word;
begin
Label1.Caption:=DateToStr(Date);
Label2.Caption:=TimeToStr(Time);
Label3.Caption:=FormatDateTime('Минское время h час. m мин.',Time);
DT:=Now;
Label4.Caption:=FormatDateTime('dd\mm\yy',DT);
Label5.Caption:=' День №'+IntToStr(DayOfWeek(Date));
DecodeDate(DT,y,m,d);
Label6.Caption:=IntToStr(y)+'.'+IntToStr(m)+'.'+IntToStr(d);
DecodeTime(DT,hh,mm,ss,ms);
Label7.Caption:=IntToStr(hh)+':'+IntToStr(mm);
end;
```



# Программа отображает на форме текущее время и дату

```
const stDay:array[1..7] of string=('воскресенье','понедельник','вторник','среда'
'четверг','пятница','суббота');
      stMonth:array[1..12] of string=('января','февраля','марта','апреля',
'мая','июня','июля','августа','сентября','октября','ноября','декабря');
```

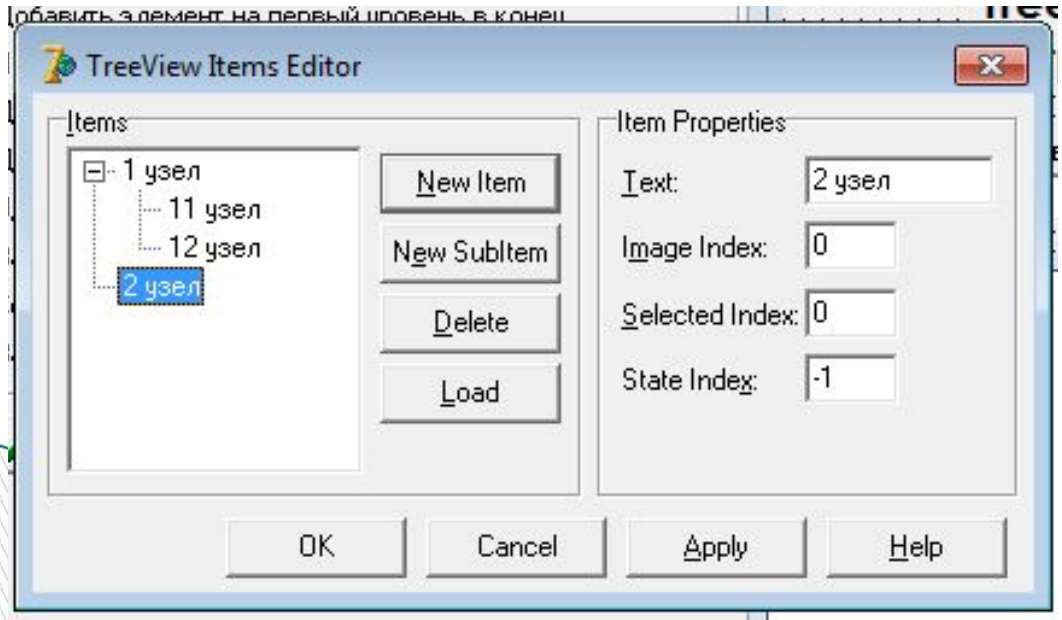
```
procedure TMainForm.Timer1Timer(Sender: TObject);
var T:TDateTime;
begin
  T:=Now;
  Time.Caption:=FormatDateTime('hh:mm:ss',T);
end;
```

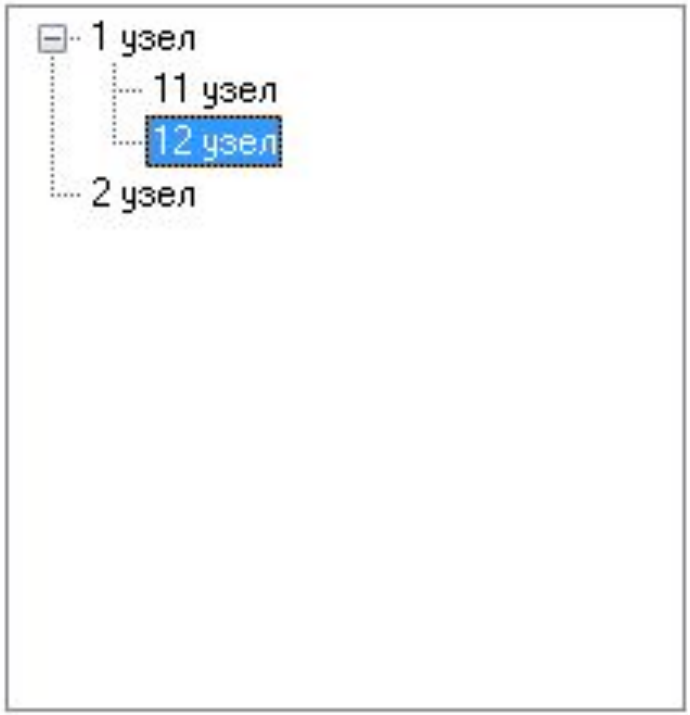
```
procedure TMainForm.FormCreate(Sender: TObject);
var Present:TDateTime;
    y,m,d:word;
begin
  Present:=Now;
  DecodeDate(Present,y,m,d);
  Date.Caption:='Сегодня '+IntToStr(d)+' '+stMonth[m]+' '
  +IntToStr(y)+' года, '+stDay[DayOfWeek(Present)];
  Timer1.Interval:=1000;
  Timer1.Enabled:=true;
end;
```

# Компонент отображения иерархических данных - TreeView

Отображает данные в виде дерева, в котором пользователь может выбирать нужный ему узел.

Во время проектирования формирование дерева осуществляется в окне редактора узлов дерева. Это окно вызывается двойным щелчком на компоненте **TreeView** или нажатием кнопки с многоточием около свойства **Items** в окне Инспектора Объектов.



# Основные свойства

***Items*** – узлы дерева. Каждый узел имеет тип TTreeNodes.
Доступ к информации об отдельных узлах осуществляется через этот индексный список узлов.
Например, **TreeView1.Items[1]** — это узел дерева с индексом 1 (второй узел дерева).
Каждый узел является объектом типа **TTreeNodes**, обладающим своими свойствами и методами.
***Selected*** – выбранный узел.
***Items.Text*** – содержание узла.

Дерево можно формировать или перестраивать и во время выполнения приложения. Для этого служит ряд методов объектов типа **TTreeNodes**. Следующие методы позволяют вставлять в дерево новые узлы:

***Items.Add(Node,s)*** – Добавляет новый узел с текстом **S** как последний узел уровня, на котором расположен **Node**.

***Items.AddFirst (Node,s)*** – Вставляет новый узел с текстом **S** как первый из узлов уровня, на котором находится **Node**. Индексы последующих узлов увеличиваются на 1.

***Items.Insert (Node,s) -*** Вставляет новый узел с текстом **S** сразу после узла **Node** на тот же уровень. Индексы последующих узлов увеличиваются на 1.

***AddChild(Node,s)*** - Добавляет узел с текстом **S** как последний дочерний узла **Node**.

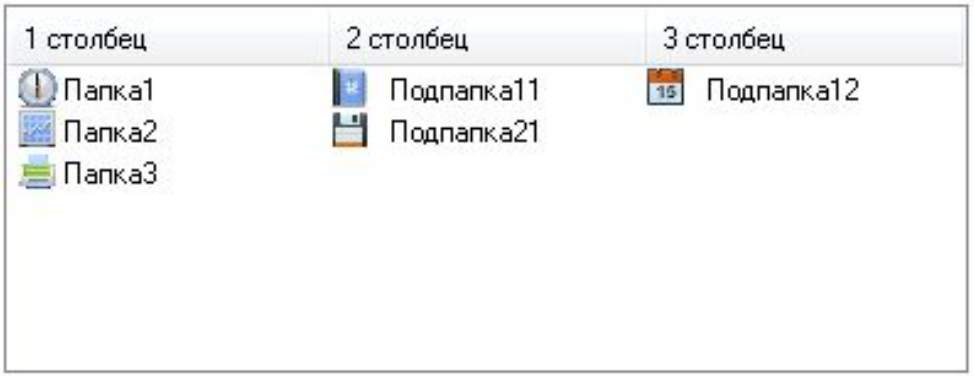
***Items.AddChildFirst(Node,s)*** – Вставляет новый узел с текстом **S** как первый из дочерних узлов узла **Node**. Индексы последующих узлов увеличиваются на 1.

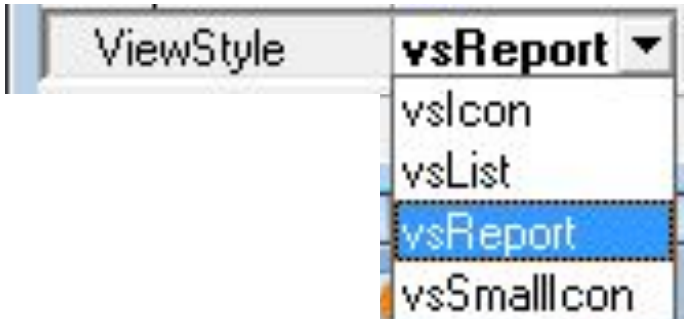
Каждый из этих методов возвращает вставленный узел.

Для удаления узлов имеется два метода: **Clear**, очищающий все дерево, и **Delete(Node: TTreeNode)**, удаляющий указанный узел **Node** и все его узлы — потомки.

# Компонент отображения иерархических данных - ListView

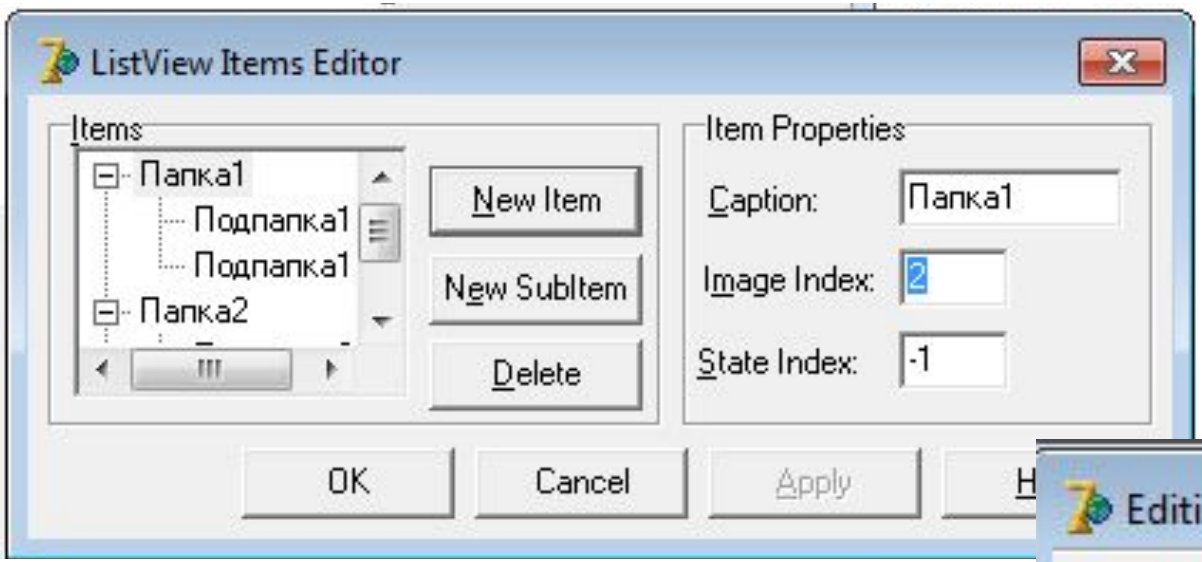
Позволяет отображать данные в виде списков, таблиц, крупных и мелких пиктограмм.



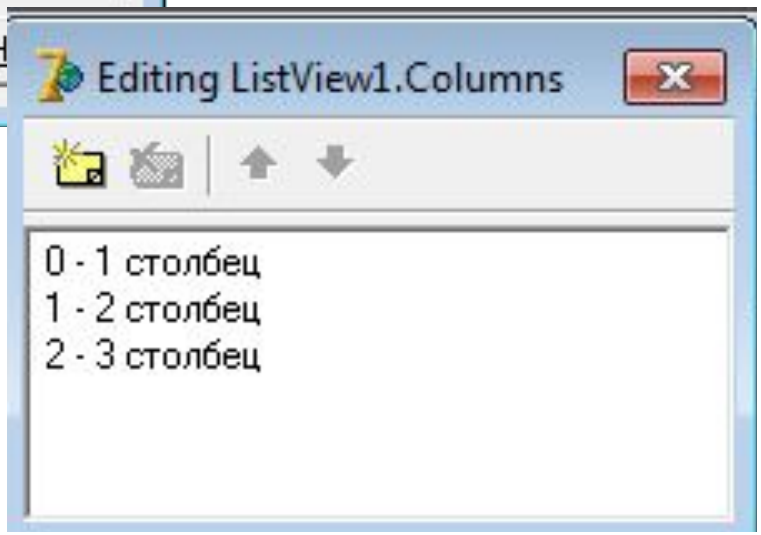





Редактор элементов списка, редактор столбцов, подключение изображений из ImageList.





# Классы и объекты. Поля, методы, свойства, события. Области видимости элементов класса

# Класс и объект

Классом называется структура языка, которая может иметь в своем составе переменные, функции и процедуры. Переменные, в зависимости от предназначения именуются полями или свойствами. Процедуры и функции класса - методами.

Например:

```
type TMyObject = class(TObject)
   MyField: Integer;
   function MyMethod: Integer;
end;
```

Поля объекта аналогичны полям записи [record]. Методы - это процедуры и функции. описанные внутри класса и предназначенные для операций над его полями. От обычных процедур и функций методы отличаются тем, что им при вызове передается указатель на тот объект, который и вызвал.

Чтобы использовать класс в программе, нужно объявить переменные этого типа.

Переменная объектного типа называется экземпляром класса или объектом:

```
var AMyObject: TMyObject;
```

Класс - это описание, объект - то, что создано в соответствии с этим описанием. Объект "появляется на свет" в результате вызова специального метода, который инициализирует объект - конструктора. Созданный объект уничтожается другим методом – деструктором.

# Поля и методы

Поля класса являются переменными, объявленными внутри класса. Они предназначены для хранения данных во время работы экземпляра класса [объекта].

Ограничений на тип полей в классе не предусмотрено. В описании класса поля должны предшествовать методам и свойствам. Обычно поля используются для обеспечения выполнения операций внутри класса.

Методом называется объявленная в классе функция или процедура, которая используется для работы с полями и свойствами класса.

Обращаться к свойствам класса можно только через его методы. От обычных процедур и функций методы отличаются тем, что им при вызове передается указатель на тот объект, который вызвал. Поэтому обрабатываться будут данные именно того объекта, который вызвал метод.

В Delphi все классы являются потомками класса "TObject". Унаследованные от класса-предка поля и методы доступны в дочернем классе; если имеет место совпадение имен методов, то говорят что они перекрываются.

В зависимости от того, какие действия происходят при вызове, методы делятся на три группы.

- Статические методы (полностью перекрываются в классах потомках при их переопределении. При этом можно полностью изменить объявление метода),
- Виртуальные [virtual] и динамические [dynamic] (при наследовании должны сохранять наименование и тип)
- Перегружаемые [overload] методы (дополняют механизм наследования возможностью использовать нужный вариант метода [собственный или родительский] в зависимости от условий применения).

Те из явлений или процессов, которые не изменяют своего содержания, должны быть реализованы в виде статических методов.

Те же, которые изменяются при переходе от общего к частному, лучше облечь в форму виртуальных методов.

Основные методы надо описать в классе-предке и затем перекрывать их в классах потомках.

В этом состоит принцип полиморфизма.

# Области видимости

- При описании нового класса важен разумный компромисс. С одной стороны, требуется скрыть от других методы и поля, представляющие собой внутренне устройство класса. Маловажные детали на уровне пользователя объекта будут бесполезны и только помешают целостности восприятия.
- С другой стороны, если слишком ограничить того, кто будет порождать классы потомки, и не обеспечить ему достаточный набор инструментальных средств и свободу маневра, то он и не станет использовать ваш класс.

В модели языка Delphi существует механизм доступа к составным частям объекта, определяющий области, где ими можно пользоваться [области видимости].

Поля и методы могут относиться к четырем группам, отличающимися областями видимости.

Методы и свойства могут быть

- общими [секция "public"],
- личными [секция "private"],
- защищенными [секция "protected"],
- опубликованными [секция "published"].

Области видимости, определяемые директивами, таковы:

- поля, свойства и методы секции "public" не имеют ограничений на видимость. Они доступны из других функций и методов объектов как в данном модуле, так и во всех прочих, ссылающихся на него;
- поля, свойства и методы, находящиеся в секции "private", доступны только в методах класса и в функциях, содержащихся в том же модуле, что и описываемый класс. Такая директива позволяет полностью скрыть детали внутренней реализации класса. Свойства и методы из секции "private" можно изменять, и это не будет сказываться а программах, работающих с объектами этого класса;

- поля, свойства и методы секции "protected" также доступны только внутри модуля с описываемым классом. Но - и это главное - они доступны в классах, являющихся потомками данного класса, в том числе и в других модулях. Такие элементы особенно необходимы для разработчиков новых компонентов-потомков уже существующих. Оставляя свободу модернизации класса, они все же скрывают детали реализации от того, кто только пользуется объектами этого класса;

- область видимости, определяемая четвертой директивой - "published", имеет особое значение для интерфейса визуального проектирования Delphi. В этой секции должны быть собраны те свойства объекта, которые будут видны не только во время исполнения приложения, но и из среды разработки. Все свойства компонентов доступны через "Инспектор объектов", являются их опубликованными свойствами. Во время выполнения такие свойства общедоступны как и "public".

# Всё!