Цикл презентаций «ООП на Delphi» посвящен объектно – ориентированному программированию с использованием одной из самых распространенных систем быстрой разработки приложений – Delphi

Используя данный учебный курс, можно самостоятельно овладеть основами объектно – ориентированного программирования на Delphi. Для расширения Ваших знаний к курсу приложен ряд учебных пособий и справочников по Delphi

Цикл содержит 13 презентаций:

ООП на Delphi – 1: Знакомство с системой программирования Borland Delphi. Объекты (компоненты) и их свойства и методы

ООП на Delphi – 2: Первая программа на Delphi, сохранение и компиляция

ООП на Delphi – 3: Программное изменение свойств объектов

ООП на Delphi – 4: Условия в Delphi. Создание простого теста

ООП на Delphi – 5: Элементы ввода и вывода информации. Обработка исключений

ООП на Delphi – 6: Заставка программы и элемент таймер

ООП на Delphi – 7: Программируем свою игрушку

ООП на Delphi – 8: Меню программы, панель статуса, диалоги

ООП на Delphi – 9: Создаем свой текстовый редактор

ООП на Delphi – 10: Базы данных на Delphi

ООП на Delphi – 11: Калькулятор на Delphi. Обработка исключительных ситуаций

ООП на Delphi – 12: Создаем тестирующую систему

ООП на Delphi – 13: Графика на Delphi

Delphi использует язык программирования Объект Паскаль, поэтому лучше сначала изучить обычный Паскаль и поработать в ТурбоПаскале, а затем и переходить к Delphi – перейти будет очень просто, т.к синтаксис языка остается неизменным.

Изучение ООП на Delphi желательно проводить в старших профильных классах – количество часов, отводимое на информатику там вполне достаточно для освоения основ ООП на Delphi

Объектно – ориентированное программирование на

Borland® DELPHI - 8

DELPHI - 8

На этом уроке:

Мы должны научиться создавать и использовать меню программы и панель статуса, а также познакомиться с диалогами

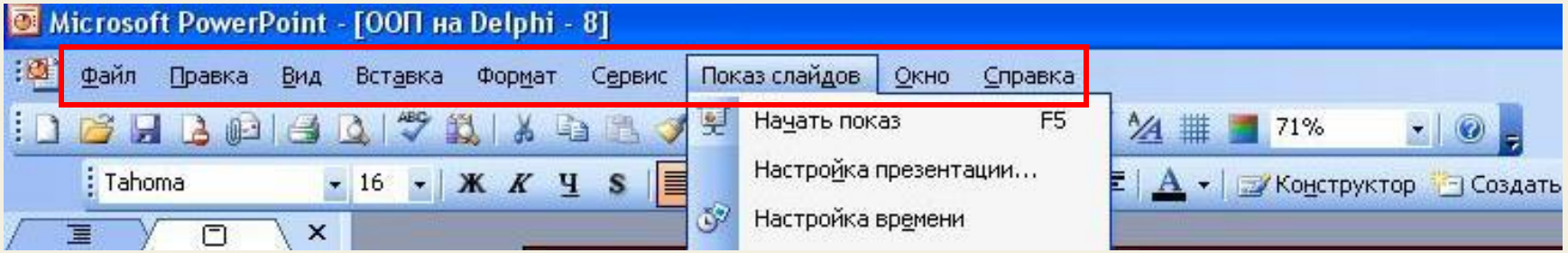
Вопросы:

1. Создание меню программы
2. Создание панели статуса
3. Использование диалогов

Borland®

Создание меню программы

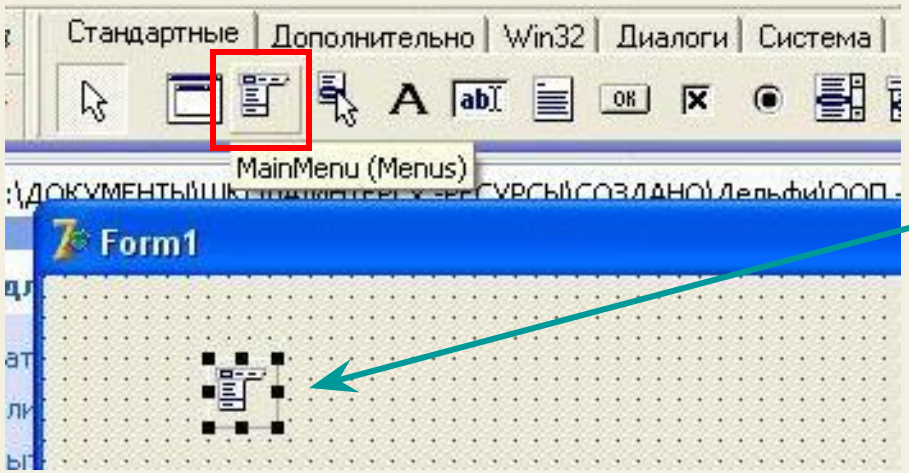
Многие профессиональные программы содержать в своей верхней части главное меню с раскрывающимися опциями (например пакет офисных программ MS OFFICE)



Система раскрывающихся меню стала своеобразным стандартом программ и является очень удобной для доступа к всем функциям программы

Рассмотрим создание такого меню:

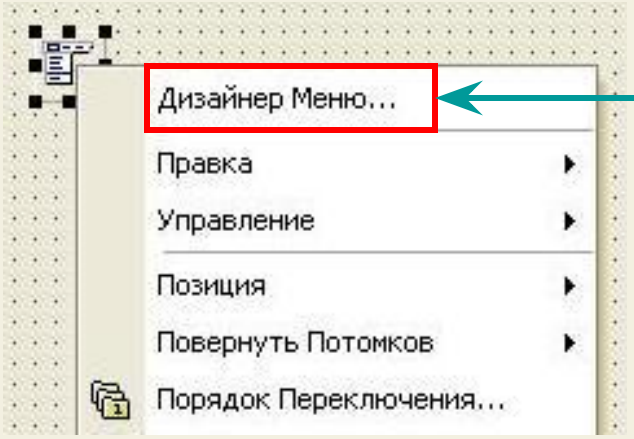
Для создания меню служит компонент Main Menu, находящийся на вкладке Стандартные



ШАГ 1

Поместим на нашу форму компонент Main Menu

ШАГ 2



Щелкнув правой по Main Menu, выберем в контекстном меню раздел Дизайнер меню – раздел, с помощью которого мы и сформируем нужное нам меню

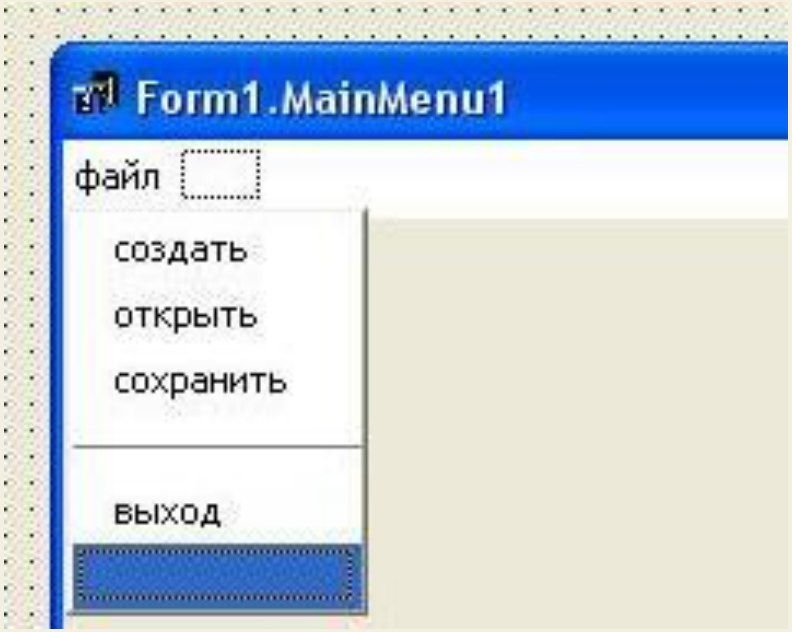
ШАГ 3

Сейчас в дизайнере меню можно сформировать нужные разделы

Сначала давайте определимся, что мы хотим иметь.

Пусть меню содержит 3 раздела:

- Файл (с опциями Создать, открыть, сохранить)
- Правка (с опциями копировать, вырезать, вставить)
- Справка (с опциями о программе, помощь)



Печатаем слово Файл и нажимаем Enter

Печатаем Создать и снова Enter, и так же открыть и сохранить

Для того чтобы создать разделяющую полосу, группирующую сходные функции, напечатаем знак «-» (минус и тоже Enter)

И так же Выход. (Наименование опций соответствует свойству Caption в инспекторе объектов)

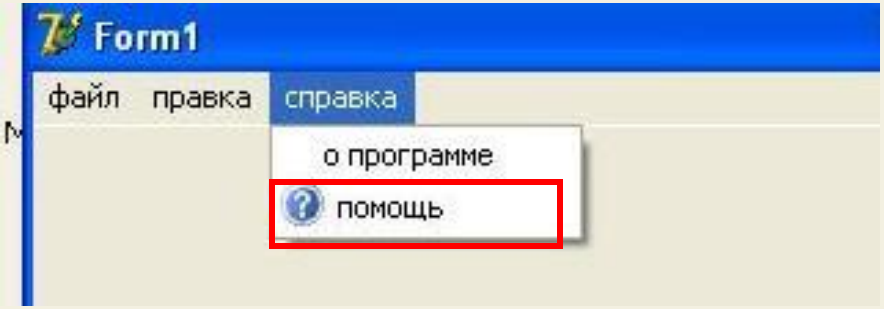
ШАГ 3

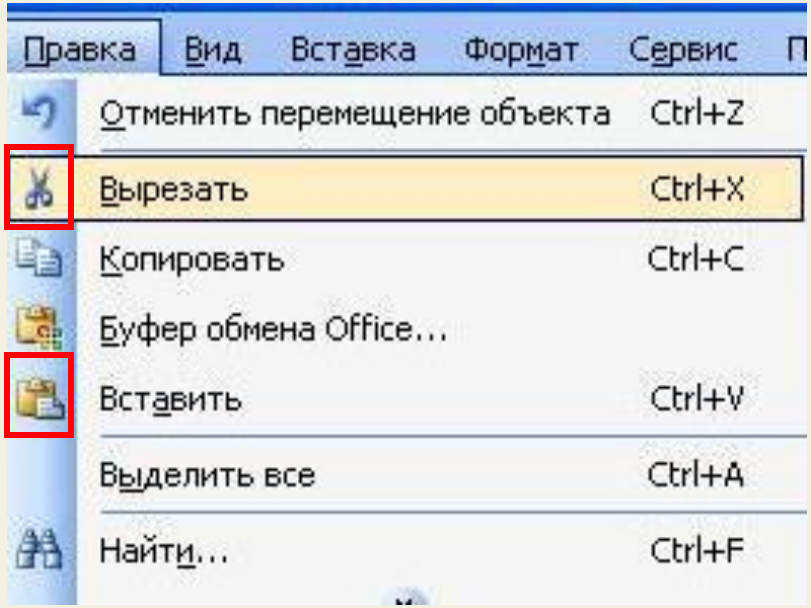
Щелкаем стрелку вправо на клавиатуре и таким же образом формируем раздел меню Правка, а затем и Справка.

Таким образом мы получили систему раскрывающихся меню

Часто раскрывающие опции имеют маленькие пиктограммы для пояснения функции данной опции

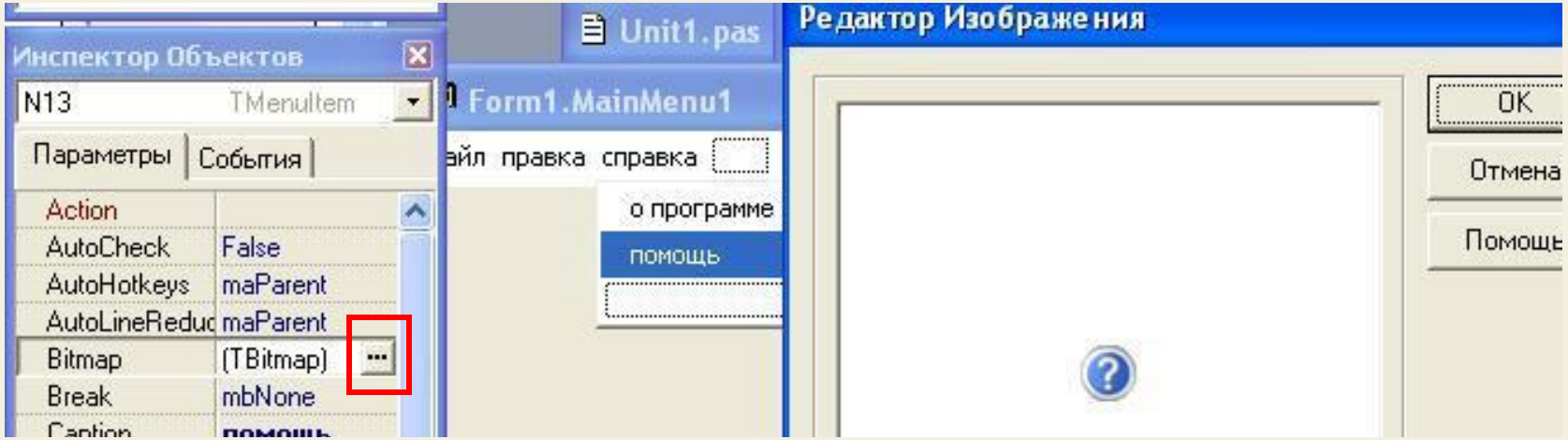
Рассмотрим, как их сделать, например для опции Помощь раздела Справка





ШАГ 3

Находясь в опции Помощь, открываем свойство Bitmap этой опции в инспекторе объектов и оказываемся в редакторе изображения, где загружаем нужную пиктограмму (алгоритм такой же, как и для командной кнопки Bit Button)



В результате мы получили систему раскрывающихся меню **Посмотреть ->**

Как видно из примера, наше меню раскрывается, но при выборе любой опции ничего не происходит т.к. мы еще не написали процедуры для обработки событий выбора опций . Этим мы займемся на следующем уроке, когда будем создавать текстовый редактор

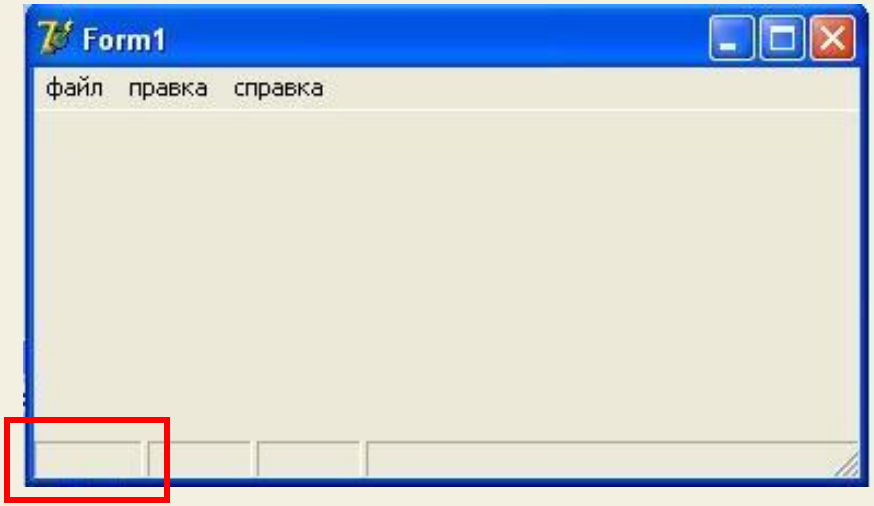
Borland®

Создание панели статуса

Панель статуса (состояния) – Status Bar, как и система раскрывающихся меню является неотъемлемой частью многих программ и располагается обычно внизу рабочего окна программы

Компонент StatusBar находится на вкладке Win 32

В нашей программе с меню уже создана панель статуса (точнее полоса состояния, состоящая из нескольких панелей (из четырех))



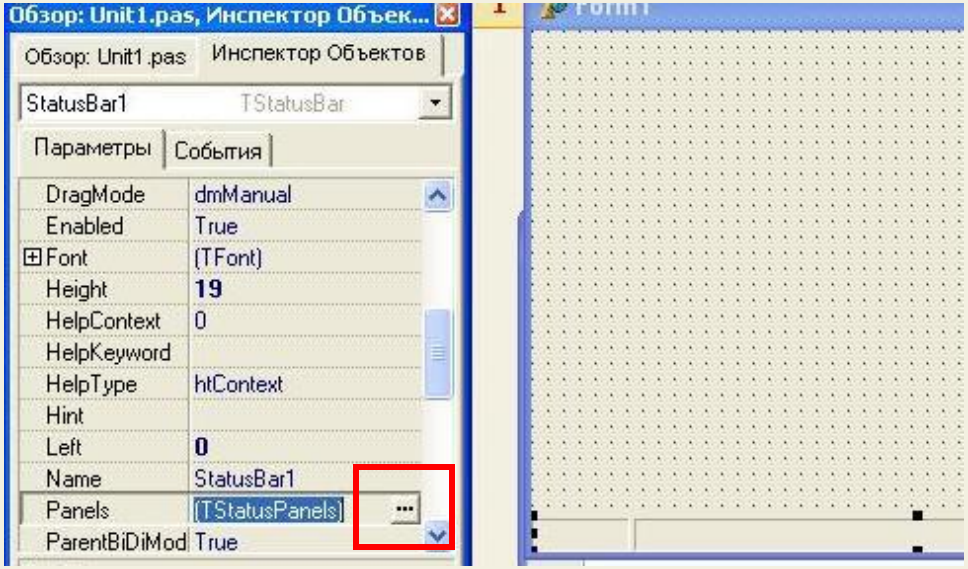
Это первая панель полосы состояния (StatusBar.Panel[0]) – нумерация панелей начинается с нуля !

Основное свойство панели – отображаемый в ней текст, например StatusBar.Panel[1].Text:='Время'; означает, что во второй панели будет выведен текст 'Время'

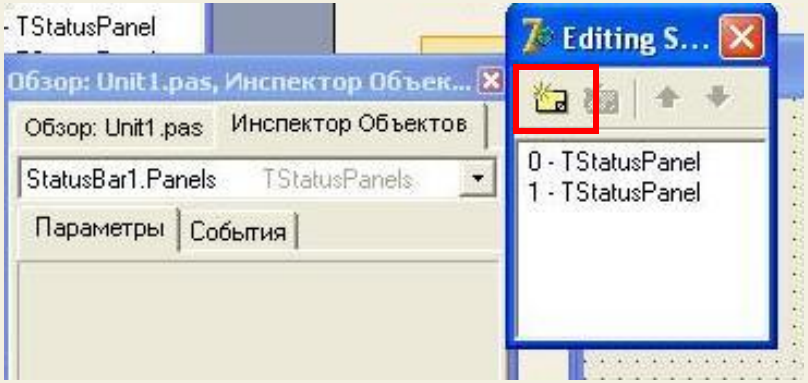
Посмотрите в инспекторе объектов другие свойства StatusBar

Для примера давайте создадим программу с полосой состояния из двух панелей, в первой из которых отображается текст «Время работы с программой :», а во второй идет отсчет времени работы с программой

ШАГ 1 Помещаем на форму компонент StatusBar



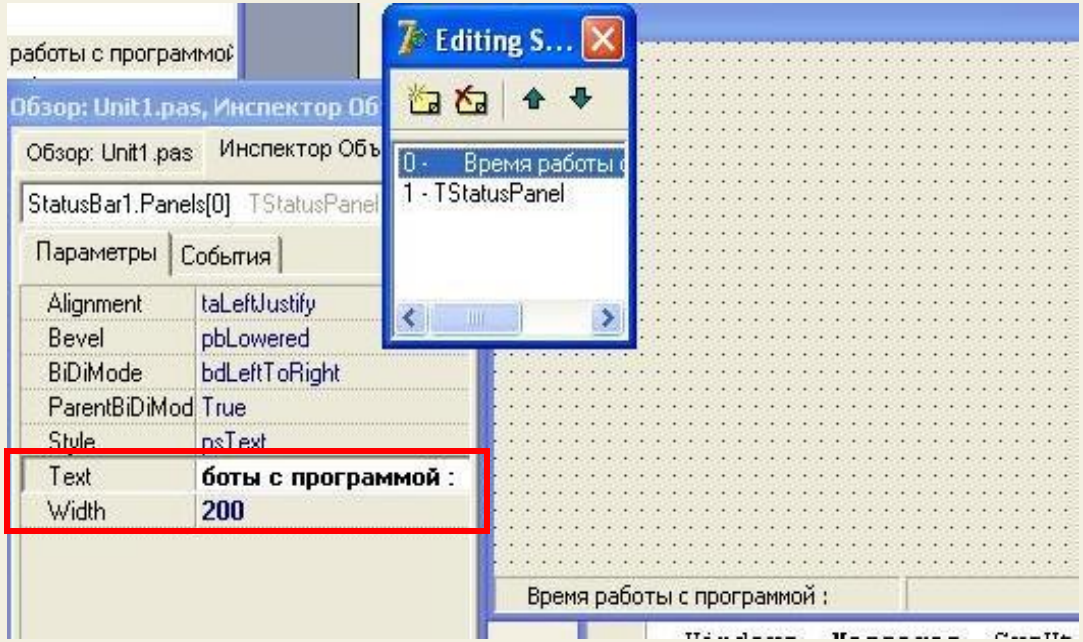
В инспекторе объектов раскрываем свойство Status Bar-а Panels



В редакторе панелей добавляем 2 панели, нажимая на пиктограмму добавления панели

Для примера давайте создадим программу с полосой состояния из двух панелей, в первой из которых отображается текст «Время работы с программой :», а во второй идет отсчет времени работы с программой

ШАГ 1 Помещаем на форму компонент StatusBar

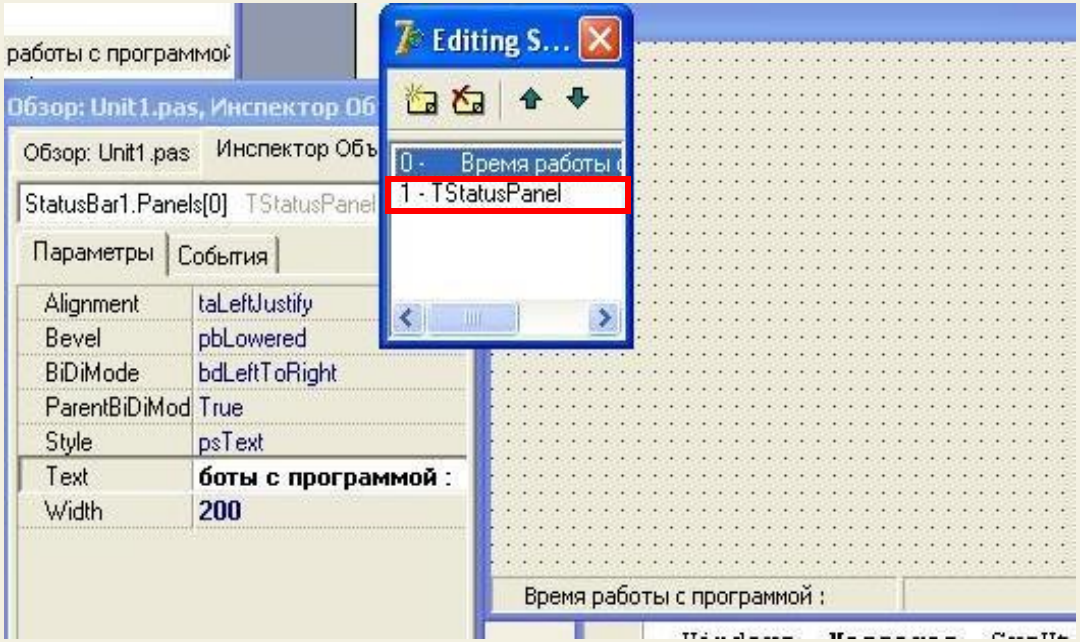


В свойстве первой панели Text (Panel 0) пишем текст «Время работы с программой»

В свойстве Width (ширина панели) поставим нужную ширину для отображения этого текста (этот текст влазит в 200 пк)

Для примера давайте создадим программу с полосой состояния из двух панелей, в первой из которых отображается текст «Время работы с программой :», а во второй идет отсчет времени работы с программой

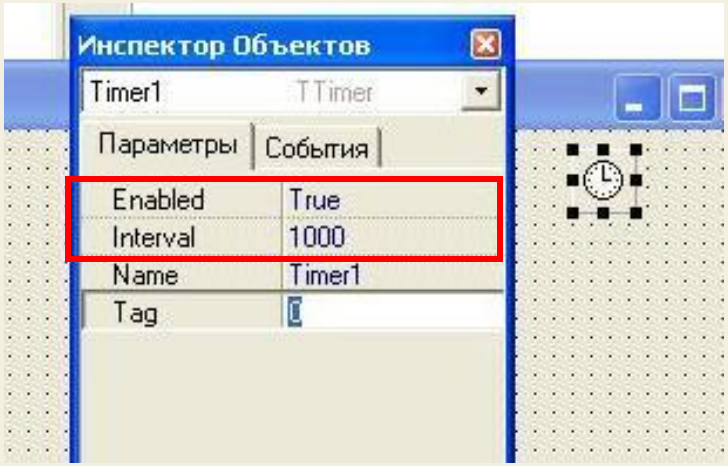
ШАГ 1 Помещаем на форму компонент StatusBar



Для второй панели (Panel 1) писать ничего не будем, потому что в ней будет идти время, поэтому сделаем это программно с использованием элемента Таймер

ШАГ 2

Поместим на форму компонент Таймер, сделаем его тикающим с частотой 1 сек (Enabled=True, Interval=1000)



При выводе в вторую панель времени работы программы нам придется использовать две переменные типа Дата/Время (TDateTime) – есть такой тип данных в Delphi, как и других системах разработки:

В первой переменной (обозначим ее S) – будет храниться время (системное время нашего компьютера) в момент старта программы и изменяться оно при работе программы не будет

Во второй переменной (обозначим ее d) – будет храниться текущее время компьютера, которое будет считываться по таймеру каждую секунду с времени операционной системы

А во второй панели (Panel 1) мы будем выводить разницу d и s, которую преобразуем из типа Дата/Время к строковому типу, соответствующему свойству Text в панели – получится время работы с программой, идущее с нуля

ШАГ 2

Рассмотрим программный код:

```
var
  Form1: TForm1;
  s,d:tdatetime;
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
s:=time;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
d:=time;

 Form1.StatusBar1.Panels[1].Text:='    '+timetostr(d-s);
end;

end.
```

Объявим переменные d и s типа Дата/Время (TDateTime)

В процедуре создания формы (запуска приложения) присвоим s текущее время (зафиксируем момент времени)

ШАГ 2

Рассмотрим программный код:

```
var
  Form1: TForm1;
   s,d:tdatetime;
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
s:=time;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
d:=time;

 Form1.StatusBar1.Panels[1].Text:='    '+timetostr(d-s);
end;

end.
```

В процедуре OnTimer с каждым «тиканьем» таймера переменной d будем присваивать текущее время операционной системы. Ясно, что значение d будет увеличиваться каждую секунду

Во второй панели выводим разницу идущего времени (d) и стоящего на месте (s), преобразуя эту разницу из формата времени (Time) в формат строки (string)

ШАГ 3

Сохраняем, компилируем и запускаем программу. Мы видим, что во второй панели идет отсчет времени работы с программой

Запускаем -> 

При создании программного кода нам часто приходится использовать различные типы данных. Также часто приходится делать преобразование типов из одних в другие. Поэтому удобно иметь под рукой справочник по работе с различными типами данных, строками, файлами, памятью – все это есть в прилагаемом к курсу справочнике «Типы данных в Delphi»

Borland®

Использование диалогов

А сейчас давайте познакомимся с организацией диалога компьютера и пользователя в процессе работы программы.

Очень часто в программах применяются диалоговые окна – для подтверждения, предупреждения, информирования пользователя – такие диалоги часто показывает нам операционная система Windows

Сейчас мы научимся создавать такие диалоговые окна

Надо сказать, что здесь мы будем использовать функции самой операционной системы Windows (WinApi)

Однако в Delphi существует большой набор компонент, реализующих стандартные функции открытия (файла), сохранения, поиска ... но с ними мы познакомимся на следующем уроке при создании собственного текстового редактора

Итак, диалоги:

Рассмотрим 3 способа организации диалогов

1. **procedure ShowMessage**
2. **function MessageDlg**
3. **function MessageBox**

---

1 способ (ShowMessage)

Самый простой способ – использование процедуры ShowMessage (показать сообщение)

Формат записи:

procedure ShowMessage(const Msg: String) – здесь const – выражение строкового типа, которое будет «выдано» в сообщении (и конечно оно записывается в апострофах). Этот способ не только самый простой, но и функционально ограниченный, т.к. в окне диалога есть только кнопка ОК, которая закрывает окно и больше никаких функций

Мы будем рассматривать диалоги на примере программы решения квадратного уравнения, созданного нами на 5 уроке, где используем все три вида диалогов для информации пользователя, обработки исключений и пр.

Давайте сначала запустим программу и посмотрим действие диалогов, а затем разберем код

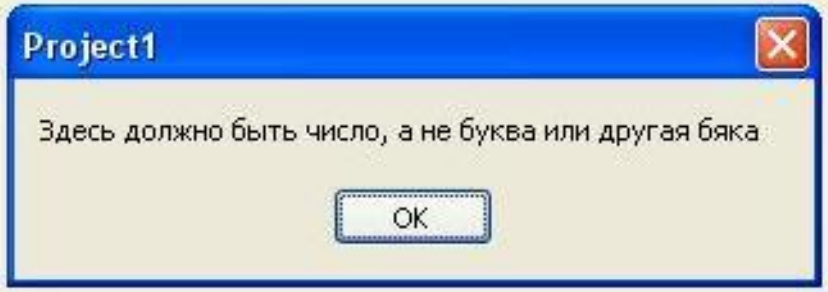
Запустите программу и попробуйте:

1. Ввести в Edit любого коэффициента не цифру, а букву
2. Нажать кнопку НАЙТИ, не введя все коэффициенты

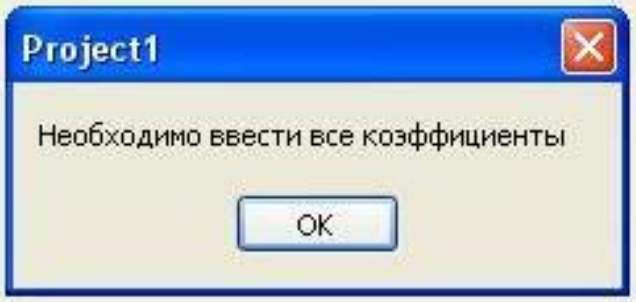
Запустить -> 

Мы видим, что выходят сообщения

- при ошибочном введении буквы вместо цифры



- при нажатии кнопки НАЙТИ, если не все коэффициенты введены



Рассмотрим код

1. Попытка ввести в Edit вместо числа букву (страховка пользователя от случайного нажатия «не той» клавиши

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
 if (Key in ['A'..'я']) or (Key in ['A'..'z'])then
  begin
Key:=#0;
ShowMessage('Здесь должно быть число, а не буква или другая бяка');
```

Здесь мы использовали событие Edit1.KeyPress (нажатие клавиши, когда фокус ввода имеет Edit1) – находим его в инспекторе объектов

А дальше понятно:

Делаем проверку условия

Если нажата клавиша, соответствующая одной из множества русских или латинских букв, то ничего не вводим (Key:=#0 – вспомните кодовую таблицу ASCII) и выдаем сообщение, что надо ввести цифру

Рассмотрим код

2. Попытка нажать кнопку НАЙТИ, когда еще не все коэффициенты введены

```
procedure TForm1.Button1Click(Sender: TObject);
begin
if (edit1.Text='') or (edit2.Text='') or (edit3.Text='') then
showmessage('Необходимо ввести все коэффициенты')
else
begin
a:=strtofloat(edit1.Text);
 b:=strtofloat(edit2.Text);
```

Здесь при нажатии кнопки НАЙТИ проверяется, а не является один из Edit-ов для ввода коэффициентов «пустым»

Если ДА, то напоминаем о необходимости ввести все коэффициенты

ИНАЧЕ следует алгоритм расчета корней уравнения ...

2 способ (MessageDlg)

function **MessageDlg** (const Msg: String; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: integer): integer,

Здесь:

const Msg: String – это текст нашего сообщения

DlgType: TMsgDlgType –это вид диалогового окна

Название параметра	**Значок**
mtWarning	
mtError	
mtInformation	
mtConfirmation	
mtCustom	Значка нет

Buttons - кнопки диалогового окна: mbYes, mbNo, MbCancel, mbRetry, mbAbort, mbOk, mbIgnore, mbHelp, mbAll, mbYesToAll, mbNoToAll - словом, все, какие ни есть и даже готовые комбинации: mbYesNoCancel, mbOKCancel, mbYesAllNoAllCancel, mbAbortRetryIgnore, bmAbortIgnore.

HelpCtx: integer – ставьте ноль

2 способ (MessageDlg)

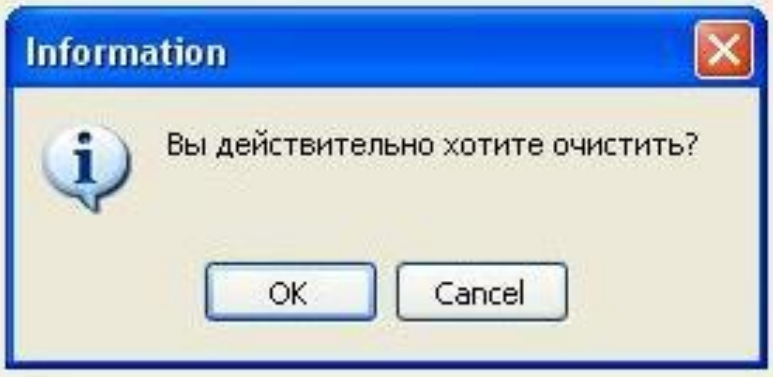
И опять давайте посмотрим пример, а затем разберем код

Запустите программу и попробуйте:

- Нажать на кнопку ОЧИСТИТЬ

Запустить -> 

В результате мы видим сообщение более функциональное



В сообщении присутствует значок для привлечения внимания и уже две кнопки, дающие возможность выбора

Недостаток: кнопки не русифицированы

Рассмотрим код

```
procedure TForm1.Button2Click(Sender: TObject);
begin
case MessageDlg('Вы действительно хотите очистить?',
mtInformation,mbOkCancel,0) of
idOk:
begin
 form1.Edit1.Text:='';
 form1.Edit2.Text:='';
  form1.Edit3.Text:='';
 form1.Edit4.Text:='';
 form1.Edit5.Text:='';
  form1.Edit6.Text:='';
end;
idCancel: Abort;
```

Это текст нашего сообщения

Ноль означает отказ от справки

Здесь мы выбрали 2 кнопки: OK и Cancel

Это вид значка в окне

Рассмотрим код

```
procedure TForm1.Button2Click(Sender: TObject);
begin
case MessageDlg('Вы действительно хотите очистить?',
mtInformation,mbOkCancel,0) of
idOk:
begin
 form1.Edit1.Text:='';
 form1.Edit2.Text:='';
  form1.Edit3.Text:='';
 form1.Edit4.Text:='';
 form1.Edit5.Text:='';
  form1.Edit6.Text:='';
end;
idCancel: Abort;
```

В этом алгоритме мы используем **case** – выбор (вспомните Паскаль)

Если нажата кнопка ОК, то очищаем все Edit -ы

Если нажата кнопка Cancel, закрываем диалог

3 способ (MessageBox)

function **MessageBox** (Parent: HWnd; Txt, Caption: PChar; TextType: Word): Integer

Здесь:

Parent: HWnd– "хозяин" (окно, владеющее сообщением)

Txt - 'текст сообщения'

Caption – заголовок диалогового окна

TextType – параметр, определяющий вид иконки в окне и какие в нем будут кнопки, например (некоторые из вариантов):

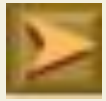
Значение TextType	**Вид иконки**	**Кнопки**
0	-	OK
1	-	OK, Отмена
2	-	Стоп, Повтор, Пропустить
3	-	Да, Нет, Отмена
19	Error	Да, Нет, Отмена
34	Confirmation	Стоп, Повтор, Пропустить
36	Confirmation	Да, Нет
51	Warning	Да, Нет, Отмена
52	Warning	Да, Нет
53	Warning	Повтор, Отмена
66	Information	Стоп, Повтор, Пропустить

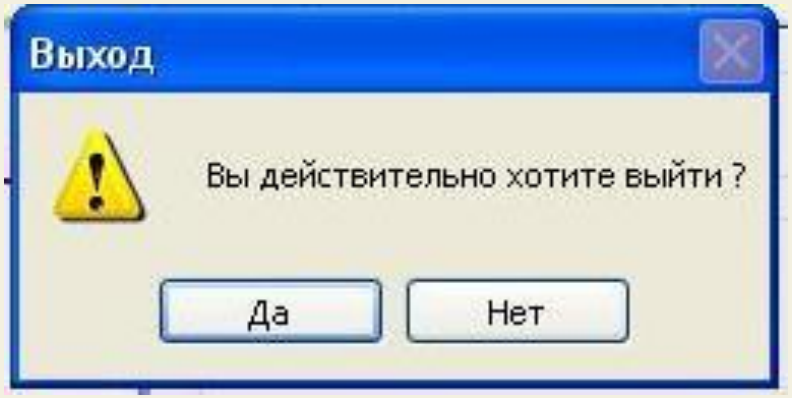
3 способ (MessageBox)

Преимущества этого способа в том, что кнопки окна «русские» (точнее говоря – они соответствуют языковой версии установленной на компьютере Windows)

Запустите программу и попробуйте:

- Нажать на кнопку ВЫХОД

Запустить -> 



В результате мы видим сообщение с русскими кнопками – это удобнее

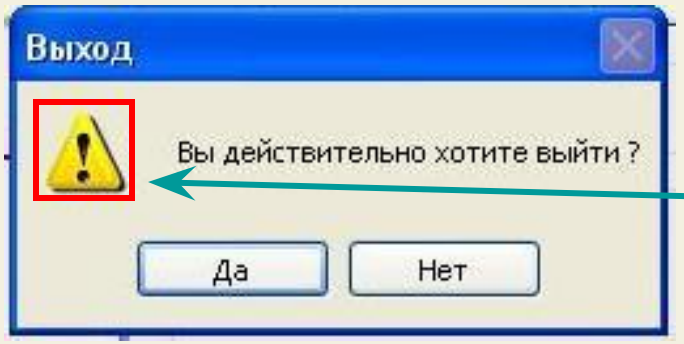
Рассмотрим код

```
procedure TForm1.Button3Click(Sender: TObject);
begin
 case MessageBox(form1.Handle, 'Вы действительно хотите выйти ?',
'Выход',52) of
id_yes:form1.Close;
id_no: Abort;
   end;
   end;
end.
```

Это текст нашего сообщения

Указываем, что «хозяином» диалога является форма Form1

Это заголовок диалогового окна



Это параметр, указывающий, что в окне 2 кнопки (ДА и НЕТ) и иконка предупреждения

52	Warning	Да, Нет

Рассмотрим код

```
procedure TForm1.Button3Click(Sender: TObject);
begin
 case MessageBox(form1.Handle,'Вы действительно хотите выйти ?',
 'Выход',52) of
id_yes:form1.Close;
id_no: Abort;
   end;
   end;
end.
```

Если выбрана кнопка ДА (Yes), то форма 1 закрывается

Если выбрана кнопка НЕТ (no), диалог прерывается

И на этом мы закончим знакомство с диалогами

ИТОГИ УРОКА:

На этом уроке мы научились создавать выпадающее меню программы, использовать панель статуса, а также познакомились с созданием диалогов пользователя и программы

НА СЛЕДУЮЩЕМ УРОКЕ:

ООП на Delphi – 9:

Мы познакомимся с стандартными диалогами и создадим свой текстовый редактор

Домнин Константин Михайлович
E – mail: kdomnin@list.ru
2006 год.