Лабораторная работа 12. Многооконные приложения. SDI- и MDI-интерфейсы

Рассматривается создание многооконных приложений. На практике научимся создавать модальные и немодальные окна. Подробно изучим создание SDI-интерфейсов, немного коснемся принципов работы MDI-интерфейсов.

Цель занятия

Научиться создавать и отлаживать многооконные приложения, применять модальные и немодальные окна.

Задание

1. Изучите изложенный материал.
2. Дополните созданное на предыдущем занятии приложение дополнительным окном.
3. Проведите тестирование полученного приложения и совместной работы нескольких окон.

Многооконные приложения

До сих пор мы с вами все приложения делали с одним единственным окном. А между тем, в современном программировании редко встречаются программы, имеющие только одно окно. Даже простые стандартные утилиты, вроде Калькулятора calc.exe или игры "Сапер" - winmine.exe имеют по нескольку окон.

Имеется два типа интерфейсов: **SDI** (**Single Document Interface** - однодокументный интерфейс) и **MDI** (**Multi Document Interface** - многодокументный интерфейс). SDI-приложения работают одновременно с одним документом, MDI-приложения предназначены для одновременной работы со множеством однотипных документов. При этом все документы располагаются внутри одного контейнера, которым служит, как правило, главная форма. Компания Microsoft не рекомендует использовать MDI-интерфейсы, хотя сама использует их в различных служебных программах, например, в консолях вроде Диспетчера устройств. Кроме того, разработка MDI-приложений в Lazarus пока не реализована, так что подробно рассматривать MDI-интерфейсы мы не будем, хотя вкратце и коснемся этой темы. Но вначале - SDI.

SDI

В SDI-приложениях окна могут быть двух видов - **модальные** и **немодальные**. Создаются они одинаково, разница заключается только в способе вывода этих окон на экран. Модальное окно блокирует программу, не даёт с ней работать, пока вы это окно не закроете. Стандартный пример модального окна - окно "**О программе**", которое присутствует почти в любом приложении. Как правило, такое окно находится в меню "**Справка**". Пока вы не нажмете "**ОК**", закрыв это окно, вы не сможете работать с основной программой.

Немодальные окна позволяют переключаться между ними, и программой, и работать одновременно и там, и там. Типичный пример - окна **Lazarus** - вы можете переключаться между **Инспектором объектов**, **Редактором кода**, **Редактором форм**, и другими окнами - они не мешают друг другу, так как все они немодальные.

Изучим работу с различными окнами на примерах.

Модальные окна

На прошлом практическом занятии мы отлаживали приложение - Блокнот-шифровальщик. Там в меню "**Справка**" было предусмотрено подменю "**О программе**", но само окно не реализовано. Пришло время исправить это упущение. Вы должны были сохранить проект в папку **12-01** под именем **CodeBook**. Убедитесь, что **Lazarus** закрыт, и загрузите файл **CodeBook.lpi** - это информационный файл проекта. В результате, загрузится Lazarus с этим проектом (с тем же успехом можно было бы загрузить файл **CodeBook.lpr**).

Выберите команду меню "**Файл -> Создать форму**" или нажмите одноименную кнопку на **Панели инструментов**. Появится новая форма с именем по умолчанию **Form1**. Для упрощения работы рекомендуется называть все формы понятными именами, и в начале имени ставить префикс `f`, что означает форму. Поэтому в свойстве `Name` этой формы напишите `fAbout`, затем нажмите кнопку "**Сохранить все**" (или выберите "**Файл -> Сохранить все**"), и модулю этого нового окна дайте имя `About`. Переключитесь клавишей **<F12>** в **Редактор кода** - вы увидите вкладки модулей:

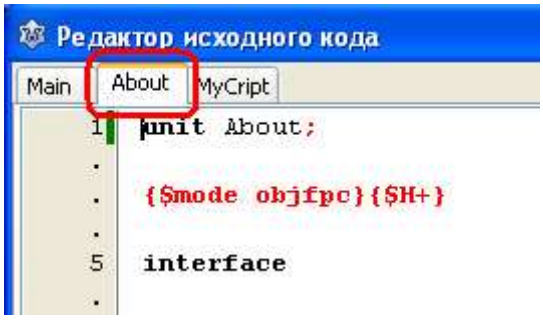

Рис. 12.1. Вкладки модулей в Редакторе кода

Два из этих модулей - `Main` и `About` имеют формы. Переходя по этим вкладкам можно переключаться между модулями. Но нам сначала нужно сделать само окно "**О программе**". Так что клавишей **<F12>** переключитесь обратно в **Редактор форм**. Прежде всего, в свойстве `BorderStyle` формы **fAbout** выберите значение `bsDialog`, так как нам не нужно, чтобы пользователь имел возможность изменять размеры окна, разворачивать или сворачивать его. Затем в свойстве `Position` выберите `poMainFormCenter`, чтобы окно появлялось по центру главного окна. До сих пор мы не устанавливали это значение у окон, так как все наши приложения содержали единственное, оно же главное окно. Окно "**О программе**" - не главное, поэтому его желательно выводить на экран по центру главного окна. Главным в проекте считается окно, созданное первым, его мы обычно называем **fMain**.

В свойстве `Caption` формы напишите "**О программе**".

Установите на форму простую панель **TPanel**, очистите ее свойство `Caption`. Чтобы сделать из панели красивую рамку, установите в её свойстве `BevelInner` значение `bvLowered`.

Далее, на панель установите две метки `TLabel` и один `TMemo`. В `TMemo` мы будем выводить многострочный текст с пояснением о назначении программы. Поскольку нам не нужно, чтобы пользователь мог редактировать этот текст, свойство `ReadOnly` компонента `Memo1` установите в `True`. Текст в `Memo1` придется вводить встроенным **Редактором** через свойство `Lines`.

Ниже панели установите кнопку **TBitBtn** с вкладки **Additional Палитры компонентов**, в свойстве `Kind` кнопки выберите значение `bkOK`.

Для экономии места я не буду подробно расписывать, как вводить в метки текст, менять у компонентов шрифты и размеры - вы прекрасно должны уметь делать это сами. В результате у вас должна получиться примерно такая форма:

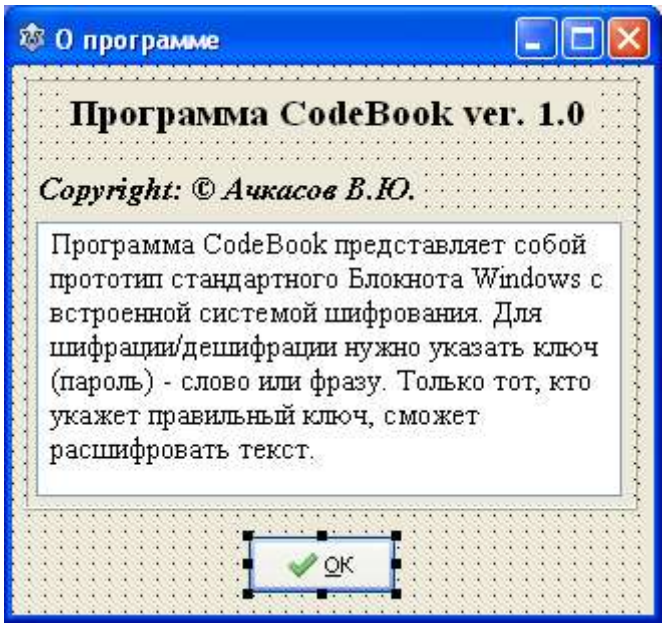

Рис. 12.2. Форма fAbout

Во всех компонентах выбран шрифт - Times New Roman - вы же можете выбрать свой, только подберите подходящие размер и начертание.

Отдельно остановлюсь на строчке `Copyright`. Слово Копирайт (англ. *Copyright*) означает авторское право. Причем авторское право может быть двух видов - имущественное и неимущественное. Если вы делаете программу на заказ, то имущественное авторское право принадлежит заказчику - он может устанавливать эту программу на сколько угодно компьютеров, продавать или дарить ее. Неимущественное право в любом случае принадлежит автору программы, то есть, вам. Оно подразумевает, что программу нельзя переименовывать или изменять её код без вашего согласия, и что в программе вы обязательно должны упоминаться, как автор. Таким образом, если вы делаете программу на заказ, вы не обязаны вместе с программой отдавать исходный код вашего проекта! Иначе получится, что вы передаете заказчику не только имущественное, но и неимущественное право, а это уже будет цена продукта на порядок выше.

Так вот, в строчке `Copyright` указывается имущественный правообладатель. Если вы делаете программу на заказ, здесь вы должны указать заказчика. Себя же вы можете упомянуть строчкой ниже, установив еще одну метку, и начав ее текст, как "**Автор:** ". Но поскольку в данном проекте заказчика у нас нет, то все авторские права принадлежат нам. Указывайте свою фамилию.

Знак авторского права © имеет в таблице символов код 0169. Чтобы вставить его в `Caption` метки, при вводе текста нажмите **<Alt>**, и удерживая его, наберите 0169. Затем отпустите **<Alt>**. Символ должен появиться в метке. Вместо этого знака иногда указывают упрощенный вариант: "`(c)`". Сделайте, как считаете нужным.

Окно мы сделали, нужно теперь научить программу выводить его по требованию пользователя. Кнопкой **<F12>** перейдите в **Редактор кода**, затем, щелкнув по вкладке **Main**, перейдите на модуль главной формы. Вот так, сходу, мы ещё не сможем вызвать форму **fAbout**, сначала нужно подключить её модуль к главной форме. В

разделе uses главной формы, после модуля MyCript через запятую добавьте модуль новой формы About. Теперь мы сможем вызывать это окно!

Сгенерируйте событие OnClick для команды меню "**Справка -> О программе**" (если вы еще помните, для этого достаточно просто выбрать данную команду). Её код очень простой:

```
procedure TfMain.HelpAboutClick(Sender: TObject);
begin
  fAbout.ShowModal;
end;
```

Метод ShowModal, указанный в коде, вызывает на экран окно **fAbout** в модальном режиме. Пока окно не закроется, с программой работать будет нельзя. Как только оно закроется, управление передастся обратно в программу. Сохраните проект, запустите его на выполнение и убедитесь, что окно "**О программе**" вызывается по требованию пользователя и закрывается кнопкой "**ОК**". Однако не спешите закрывать проект, он нам еще понадобится.

Немодальные окна

Как уже упоминалось, немодальные окна могут быть открыты одновременно с основной программой, и не мешают её работе. Чтобы продемонстрировать работу немодальных окон, снабдим наш проект **Блокнота-шифровальщика** еще одной полезной функцией - статистикой. Создадим еще одно маленькое окно, в котором будем выводить количество строк в тексте, количество слов и символов.

Создайте новую форму командой "**Файл -> Создать форму**" или одноименной кнопкой на панели инструментов Lazarus. Форму (свойство Name) назовите **fStats**, а ее модуль сохраните под именем Stats. Далее, в свойстве Caption формы напишите "Статистика". Свойство BorderStyle установим в bsDialog, чтобы пользователь не мог менять размеры формы, а Position - в poMainFormCenter, чтобы окно появлялось по центру главного окна.

Теперь нам понадобятся шесть меток TLabel. Первые три установите в левой части формы, одну под другой. Эти метки мы переименовывать не будем, так как нам не придется обращаться к ним программно. В первой напишите "Строк:", во второй "Слов:" и в третьей "Символов:". Не забывайте ставить двоеточие после этих слов. Затем в правой части, так же одну под другой, установим еще три метки, их уже переименуем. В свойстве Name этих меток напишите, соответственно, LinesCount, WordsCount и CharsCount, а в свойстве Caption всех трех меток установите "0" (ноль).

В центре нижней части формы с вкладки **Additional Палитры компонентов** установите кнопку **TBitBtn**, в свойстве Kind которой выберите значение bkOK. В результате, у вас должна получиться примерно такая форма:

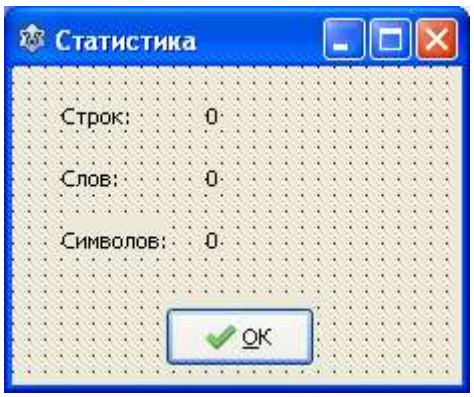

Рис. 12.3. Форма fStats

Сгенерируйте следующее событие `OnClick` для кнопки "**ОК**":

```
procedure TfStats.BitBtn1Click(Sender: TObject);
begin
  Close;
end;
```

То есть, когда пользователь нажмет на эту кнопку, окно статистики закроется.

Далее перейдем в модуль главного окна **fMain**. Чтобы мы могли оттуда вызвать новое окно, нужно через запятую добавить его модуль `Stats` в конец раздела `uses`, как это мы делали с модулем `About` в прошлом примере. Сюда же добавьте еще два модуля: `LCLProc ИЛИ LazUTF8` и `StrUtils`. В первом реализованы UTF8-функции, одна из них нам понадобится для определения количества символов. Во втором реализовано множество полезных строковых функций, включая и ту, которая нам нужна для подсчета количества слов в тексте:

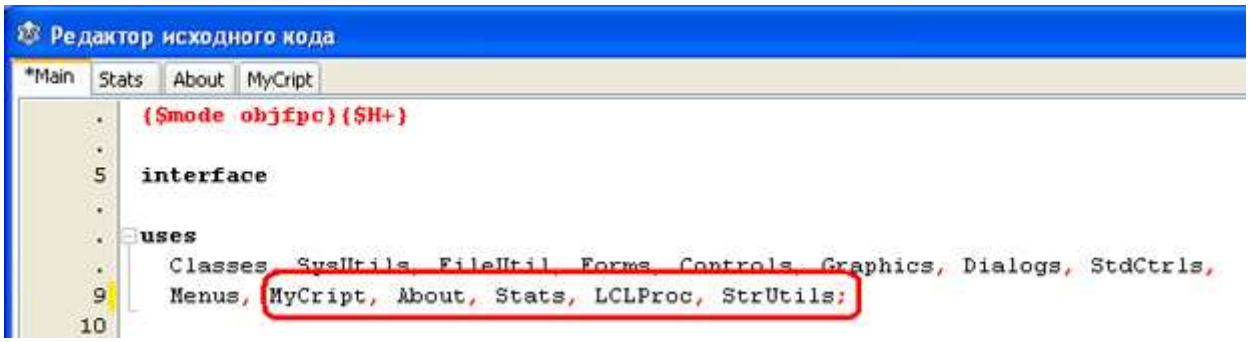

Рис. 12.4. Присоединенные модули

Теперь подумаем, каким образом мы будем вызывать окно статистики. Лучше всего поместить вызов статистики в раздел меню "**Файл**". Дважды щелкните по **MainMenu1**, чтобы вызвать редактор меню. В разделе меню "**Файл**" выделите подраздел "**Сохранить как...**", щелкните по нему правой кнопкой и выберите команду "**Вставить новый пункт (после)**". В свойстве `Name` нового подпункта введите `FileStat`, а в свойстве `Caption` напишите "`Статистика`":

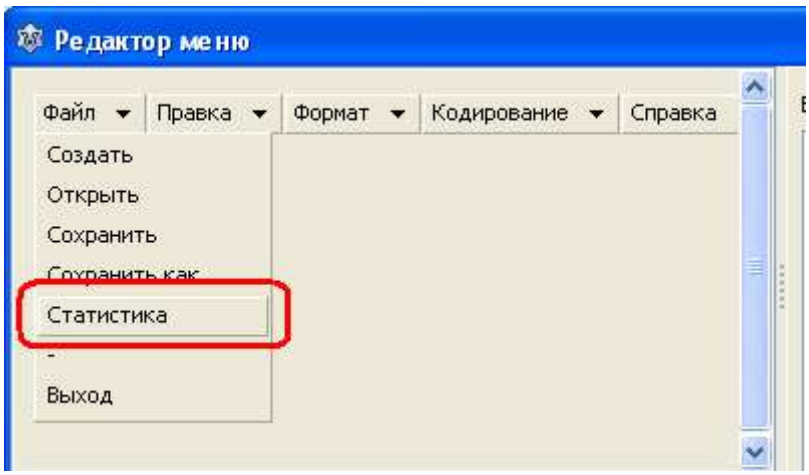

Рис. 12.5. Новый подпункт в меню

Теперь закройте **Редактор меню**; выбрав команду "**Файл -> Статистика**" сгенерируйте событие `OnClick` для нового подпункта. Код события следующий:

```
procedure TfMain.FileStatClick(Sender: TObject);
begin
  fStats.Show;
end;
```

Как видите, здесь мы вызываем форму **fStats**, но не методом `ShowModal`, как в прошлом примере, а методом `Show`. В результате окно будет показано, как немодальное. Однако этого недостаточно, чтобы окно показывало статистику. Нам нужно еще подсчитать и вывести на экран количество строк, слов и символов.

Выделите компонент `Memo1`. В Инспекторе объектов перейдите на вкладку "**События**" и сгенерируйте для него событие `OnChange`, дважды щелкнув по нему. Это событие возникает всякий раз при изменении текста в **Memo1**, тут мы и будем считать статистику. Код события следующий:

```
procedure TfMain.Memo1Change(Sender: TObject);
begin
  //считаем символы:
  fStats.CharsCount.Caption:= IntToStr(UTF8Length(Memo1.Text));
  //слова:
  fStats.WordsCount.Caption:= IntToStr(WordCount(Memo1.Text, StdWordDelims));
  //строки:
  fStats.LinesCount.Caption:= IntToStr(Memo1.Lines.Count);
end;
```

Тут для нас много нового, так что разберем код подробней. Вначале функцией `UTF8Length` мы получили количество символов в тексте `Memo1`, включая служебные символы перехода на новую строку. Это - целое число, которое нам пришлось преобразовать в строковую форму с помощью функции `IntToStr`, чтобы мы могли присвоить эту строку свойству `Caption` метки `CharsCount`. Обратите внимание, что метка находится не на этой, а на другой форме, поэтому нам пришлось сначала указать имя этой формы, затем имя метки и уж потом свойство `Caption`.

Слова в тексте считать сложнее, для этого мы воспользовались новой для нас функцией `WordCount`. Функция описана в модуле `StrUtils`, который мы подключили, она возвращает количество слов в указанном тексте, и имеет следующий синтаксис:

```
function WordCount(const S: String; const WordDelims:TSysCharSet):integer;
```

Константа `S` содержит текст, в котором подсчитываются слова, в нашем случае, это `Memo1.Text`. Константа `WordDelims` содержит список символов-разделителей, которыми может отделяться одно слово от другого. Проще всего в этом параметре воспользоваться системной константой `StdWordDelims`, которая уже описана в том же модуле, и объявлена следующим образом:

```
const StdWordDelims = [#0..' ', ',', '.', ';', '/', '\', ':', '''', '"', '`']
+ Brackets;
```

То есть, в этой константе перечислены основные символы-разделители слов. Функция возвращает целое число - количество слов в указанном тексте. Преобразовав его в символьную форму, мы присваиваем это значение свойству `Caption` метки `WordsCount`.

А вот количество строк даже считать не нужно, оно содержится в свойстве `Memo1.Lines.Count`. Нам остается лишь преобразовать его в символьную форму, и присвоить соответствующей метке. Теперь каждый раз, как в `Memo1` изменится текст (пользователь ввел символ, скопировал текст в `Memo` или удалил текст), это событие будет пересчитывать количество слов, символов и строк, и выводить их в форму **fStats**, даже если ее не видно. Выбрав команду меню "**Файл -> Статистика**", пользователь сможет вывести окно статистики. Более того, он может, не закрывая этого окна, вернуться в главную форму и продолжать набирать текст. Окно статистики при этом будет параллельно отображать результаты:

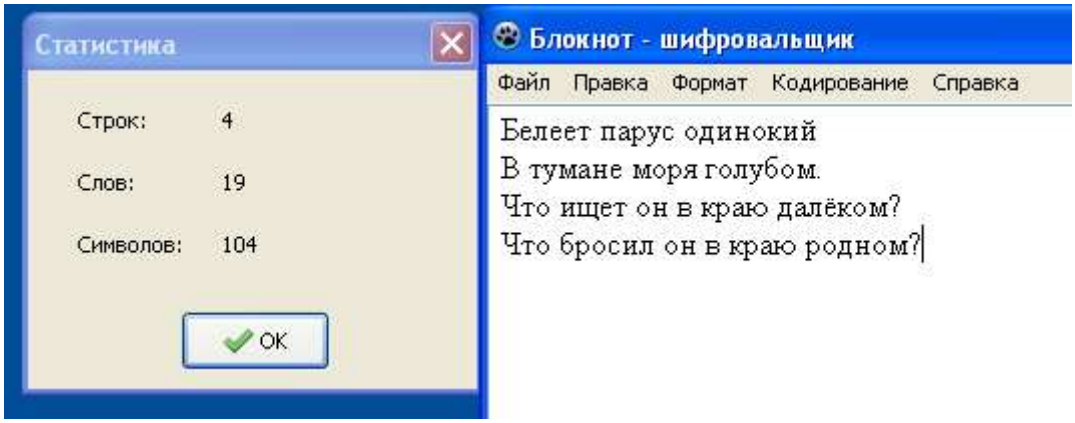

Рис. 12.6. Немодальное окно статистики в действии

Сохраните проект, запустите его на выполнение и убедитесь, что окно статистики не мешает работе главного окна. Проект **Блокнота-шифровальщика** нам потребуется ещё в лекциях №№ 28 и 29, так что не удаляйте его.

MDI-приложения

В отличие от Delphi, в Lazarus пока не реализована возможность создания MDI-приложений, а поскольку MDI-интерфейсы считаются устаревшими, то возможно, она и не будет реализована. Но знать об этих интерфейсах нужно, поэтому вкратце коснемся

этой темы, тем более, что я могу ошибаться, и в следующих версиях Lazarus разработчики эту возможность все же реализуют. Принцип создания MDI-приложений следующий:

1. Вначале вы создаете главное, оно же родительское окно. Это окно будет служить своеобразным контейнером для дочерних окон, поэтому основная, рабочая часть главной формы должна быть свободной. В свойстве `FormStyle` (стиль формы) родительского окна следует выбрать значение `fsMDIForm`.
2. Затем вы создаете дочернее окно. Во время работы программы дочернее окно будет создаваться внутри родительского, и не сможет покинуть его пределы. В свойстве `FormStyle` дочернего окна следует выбрать значение `fsMDIChild`.
3. Форму дочернего окна мы конструируем только один раз, но в программе это окно можно вызывать сколько угодно много раз, открывая в нем разные документы. Все эти дочерние окна могут быть открыты одновременно.
4. MDI-интерфейс приложения не запрещает вам создавать также модальные и немодальные окна. Например, в MDI-приложении вы с таким же успехом и таким же образом можете создать окно "**О программе**". Если вы будете создавать отдельные модальные и немодальные окна, то в свойстве `FormStyle` этих форм следует оставить значение по умолчанию `fsNormal`.

Для реализации MDI-интерфейса сами разработчики рекомендуют установить дополнительный компонент `MultiDoc`, который реализует псевдоMDI-интерфейс. Рассматривать работу с нестандартными компонентами мы не будем, желающих отсылаю на сайт разработчиков по адресу: http://wiki.freepascal.org/MultiDoc

Русскоязычной страницы этого компонента, к сожалению, нет, так что вам придется воспользоваться каким-нибудь переводчиком, например, **Promt**.