Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования

Ивановский государственный химико-технологический университет

Технология объектного программирования

Часть 2

Лабораторный практикум

Составители: Е. К. Шигалов
В. А. Бобкова

Иваново 2008

ŋqdqɏeɎ bɆ dədɆbɎɜə d ɕɔɑbɔɏ Ɉqddɏqɏè

Составители: Е.К. Шигалов, В.А. Бобкова

УДК 613.19

Технология объектного программирования. Часть 2: Лабораторный практикум. / Сост.: Е. К. Шигалов, В. А. Бобкова; ГОУВПО Иван. гос. хим.-технол. ун-т. Иваново, 2008. 72 с.

В учебном пособии представлены материалы для выполнения лабораторного практикума по курсу «Технология объектного программирования». Пособие имеет целью поддержку второй части единого курса по указанной дисциплине. Рассмотрены следующие нетрадиционные приемы: использование технологии файлов, отображенных в память, для корректировки дисковых файлов; консольные приложения и использование функций ShellAPI Windows; разработка собственных визуальных и невизуальных компонентов как на основе базовых классов, так и “с нуля”. Эти приемы активно используют технологию объектно-ориентированного программирования.

Пособие предназначено для самостоятельной работы студентов специальностей «Информационные системы и технологии» и «Информационное обеспечение систем управления».

Ил. 9. Библиогр.: 5 назв.

Рецензент доктор технических наук, профессор А.Н.Лабутин (Ивановский государственный химико-технологический университет)

1. Файловые потоки (FileStream). Создание объекта программным путем. Сохранение объекта в файле и считывание его из файла

Цель занятия: ознакомление с методами ReadComponent и WriteComponent класса TFileStream; изучение примера создания объекта на основе базового класса TButton программным путем.

Технология работы с файловыми потоками применяется для создания файлов, организации доступа (чтения, записи, поиска данных, навигации по файлу) к данным при обработке *типизированных файлов,* то есть файлов из записей, структуру которых определяет пользователь. Запись - это прежде всего пользовательский тип данных, некая структура. Заметим, что в некоторых языках программирования записи так и называются структурами, хотя существуют также понятия массива структур и структур в массивах. Применим запись к нестандартной задаче - сохранению некоторого объекта вместе с его свойствами и методами в дисковом файле.

Задание.

1. Расположите на форме (рис.1) три обычные кнопки с именами btn_Create, btn_Write и btn_Read, выполняющие следующие действия:
   - btn_Create создает обычную кнопку на основе класса TButton, определяя для нее один метод – реакцию на щелчок мышки, то есть обработчик события OnClick;
   - btn_Write записывает созданную кнопку в файл;
   - btn_Read восстанавливает записанную кнопку из файла, одновременно переопределяя метод обработки события OnClick. Внешне это выглядит как вывод некоторого текста в ответ на щелчок после восстановления кнопки с диска.

2. Используя методы обработки ошибок, корректно удалите файл, остающийся на диске после сохранения кнопки. Учтите, что с этим файлом в данном случае не связана никакая файловая переменная. Следовательно, файловую переменную нужно объявить, назначить файлу, файл закрыть (Close) и после этого удалить (Erase). Можно считать, что открытый файл – это ресурс, который необходимо вернуть операционной системе.

3. Просмотрите в **Help** иерархию класса **TFileStream**, уточните, какие классы его порождают и какие еще методы, кроме WriteComponent и ReadComponent, какие события и свойства класс **TFileStream** наследует из этих конкретных классов.

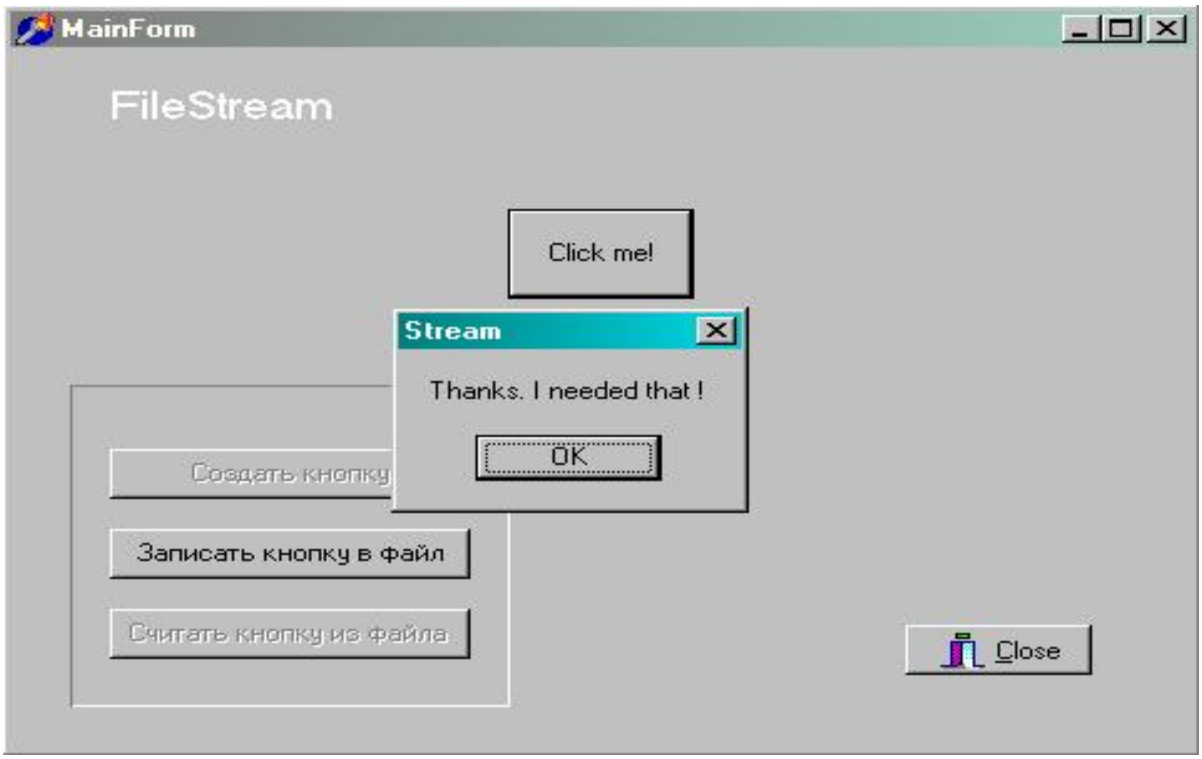

Рис.1. Вид главной формы приложения

Указания и рекомендации

В конце раздела описания класса формы введите приведенный ниже код, определяющий три процедуры в части **Private**. Первая из этих процедур будет обеспечивать доступность всех кнопок формы в зависимости от текущей ситуации, например, сделает недоступной кнопку "btn_Create" после создания тестируемой кнопки. Запустите программу **Stream.exe** для ознакомления с тем, как это может быть реализовано.

. . . . . .

```
  private
    { Private declarations }
    procedure EnableButtons;
    procedure BtnOnClick(Sender: TObject) ;
    procedure BtnOnClick2(Sender: TObject);
  public
    { Public declarations }
  end;
Const
  File_Name = 'StreamButton.str' ;
var
  MainForm: TMainForm;
  Btn: TButton;
implementation
```

Для определенности назовите файл для хранения кнопки "StreamButton.str", а создаваемую программой кнопку назовите "Btn". Это будет объект класса TButton:

```
Var
  Btn: TButton;
```

Первая из процедур (в виде заготовки) может быть подобна следующей:

```
Procedure TMainForm.EnableButtons;
{ процедура определяет доступность кнопок }
 begin
// Начальная инициализация доступности трех кнопок, как "False"
   btn_Create.Enabled := False;
   btn_Write.Enabled := False;
   btn_Read.Enabled := False;
   if Btn <> Nil  Then                  // кнопка Btn создана
     btn_Write.Enabled := True
    else
     if FileExists (File_Name) Then
       btn_Read.Enablwd:=True    { разрешение на чтение, если файл существует }
     else  btn_Create.Enabled:=True;
end;
```

Эта процедура может быть применена сразу при создании формы следующим образом:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
//  RegisterClass (TButton);   { можно и здесь }
   EnableButtons;
end;
```

а далее должна вызываться при каждом нажатии любой из кнопок и определять их доступность.

Две другие процедуры будут обработчиками одного и того же события **OnClick** для создаваемой кнопки, но назначаемыми в разное время ее жизни. В частности, эти процедуры могут выводить два разных сообщения в диалоговое окно. Они будут заимствоваться у формы. Это – методы формы, совместимые по типу с событием OnClick, то есть явно имеющие только один параметр Sender типа TObject. В реальности можно было бы для новой кнопки определить класс, для этого класса написать пользовательский метод, определяющий любой сложности реакцию на щелчок и перекрывающий (override) стандартную реакцию на Click (если бы она была запланирована в классе-предшественнике).

Приведем в качестве примера простой код этих процедур-обработчиков:

```
procedure TMainForm.BtnOnClick(Sender: TObject);
begin
  ShowMessage ('Thanks. I needed that !');
end;

procedure TMainForm.BtnOnClick2(Sender: TObject);
begin
  ShowMessage ('Hello. It "s nice to be back !');
end;
```

Для создания кнопки "в коде" используйте следующую процедуру:

```
procedure TMainForm.btn_CreateClick(Sender: TObject);
begin
 Btn :=TButton.Create (Self);
 Btn.Parent:=Self;
 Btn.Left:=200;
 btn.Top:=80;
 btn.Height:=45;
 btn.Caption:='Click me!';
 btn.OnClick := btnOnClick;          // назначили первый обработчик события
 EnableButtons;
end;
```

Процедура *для записи кнопки* имеет вид:

```
procedure TMainForm.btn_WriteClick(Sender: TObject);
 Var
  Stream : TStream;
begin
  Stream := TFileStream.Create (File_Name, fmCreate );
  try
    Stream.WriteComponent(Btn);
    Btn.Free;
    Btn:=Nil;
    EnableButtons;
  finally
    Stream.Free;
  end;
end;
```

Посмотрите в **Help**, какие значения параметра, помимо fmCreate, может иметь конструктор Create класса TFileStream (рис.2).

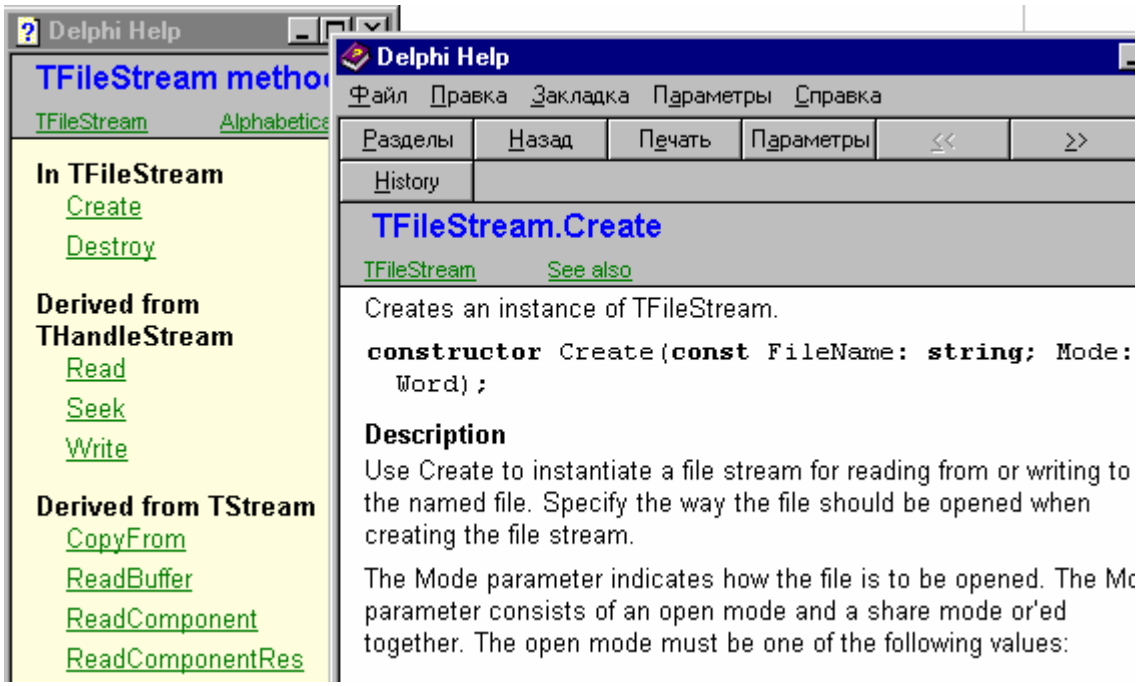

Рис.2. Описание конструктора Create класса TFileStream.

Процедура *для чтения кнопки из потока*:

```
procedure TMainForm.btn_ReadClick(Sender: TObject);
 Var
   Stream : TStream;
begin
   Stream := TFileStream.Create (File_Name, fmOpenRead );
   try
//   btn:= (Stream.ReadComponent(Nil));          // ошибка: справа - тип TComponent !
     btn:= TButton(Stream.ReadComponent(Nil)); // явное приведение типа  к TButton
     btn.Parent:=Self;          { всегда назначайте "родителя" }
     btn.OnClick := btnOnClick2;          // заменили "для интереса" обработчик
     EnableButtons;
   finally
    Stream.Free;
   end;
end;
```

В заключение заметим, что в таком виде вам не удастся сохранять кнопку в потоке и считывать ее, так как не хватает еще одного оператора а именно:

RegisterClass(TButton);

Если кнопка, или любой компонент, появилась на форме в процессе проектирования, то класс формы и все компонентные классы будут зарегистриро-

ваны автоматически. В нашем случае кнопка создается программно, поэтому ее класс должен быть зарегистрирован вручную. Только тогда экземпляры этого класса могут быть использованы (загружены или сохранены) в VCL-потоковой системе Delphi. Это может быть сделано как в начале основной программы (см. ниже), так и в методе FormCreate.

```
{ ===== Основная программа ====== }
begin
  RegisterClass (TButton);
  // ShowMessage (GetClass('TButton').ClassName);
end.
```

Информация о процедуре RegisterClass в Help выглядит таким образом:

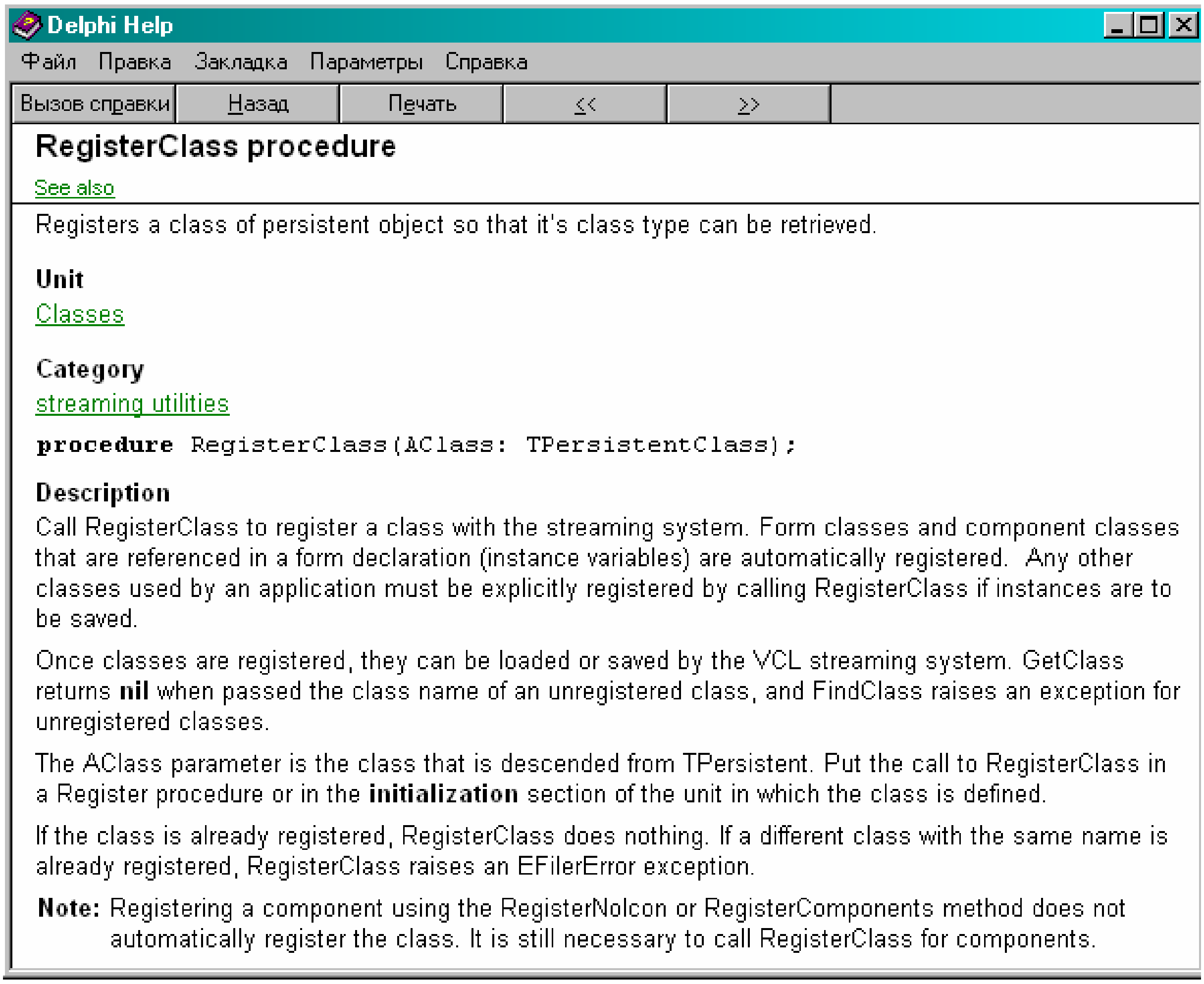

Рис.3. Описание процедуры RegisterClass.

Для закрытия формы может быть применен следующий код, использующий в целях большей надежности обработку исключений:

ŋqdqɏeɎ bɆ dədɆbɎɜə d ɕɔɑbɔɏ Ɉqdqɏqɏè

```
procedure TMainForm.FormClose(Sender: TObject;
                              |   var Action: TCloseAction);
var
  F: TextFile ;
begin
  if FileExists (File_Name) Then
    begin
//  ShowMessage ('Файл существует');
     try
       AssignFile (F,File_Name);     // 'StreamButton.Str');
       Reset(F);
       if MessageDlg('Удалить файл "' + File_Name + '" ?',
         mtConfirmation, [mbYes, mbNo], 0) = mrYes  Then
         begin
          CloseFile(F);
          Erase (F);
         end;
      except
        on EInOutError do
        MessageDlg('File I/O error.', mtError, [mbOk], 0);
     end;    // try
    end;     // begin
  end;
{ ===== Основная программа ====== }
begin
  RegisterClass (TButton);

end.
```

2. Работа с файлами, отображенными в память

Цель занятия: изучение техники отображения файлов в оперативную память и его корректировки на примере произвольного текстового файла.

Технология отображения файлов в память позволяет обойтись без обычных операций файлового ввода-вывода, а именно: связать физическую память на диске с адресом некоего виртуального адресного пространства вашей программы и сослаться на содержимое файла с помощью обычного указателя так, как будто файл располагается в оперативной памяти. Достоинство метода состоит в том, что он позволяет манипулировать огромными блоками данных, то есть открывать файлы размером в несколько гигабайт. Более того, при использовании этого метода можно отображать файлы частично. Но в нашем примере объем файла не должен быть очень большим, и файл будет текстовым, так как мы отобразим его содержимое (для визуального контроля) в редактируемом элементе управления TMemo.

Внешний вид программы представлен на рис.4.

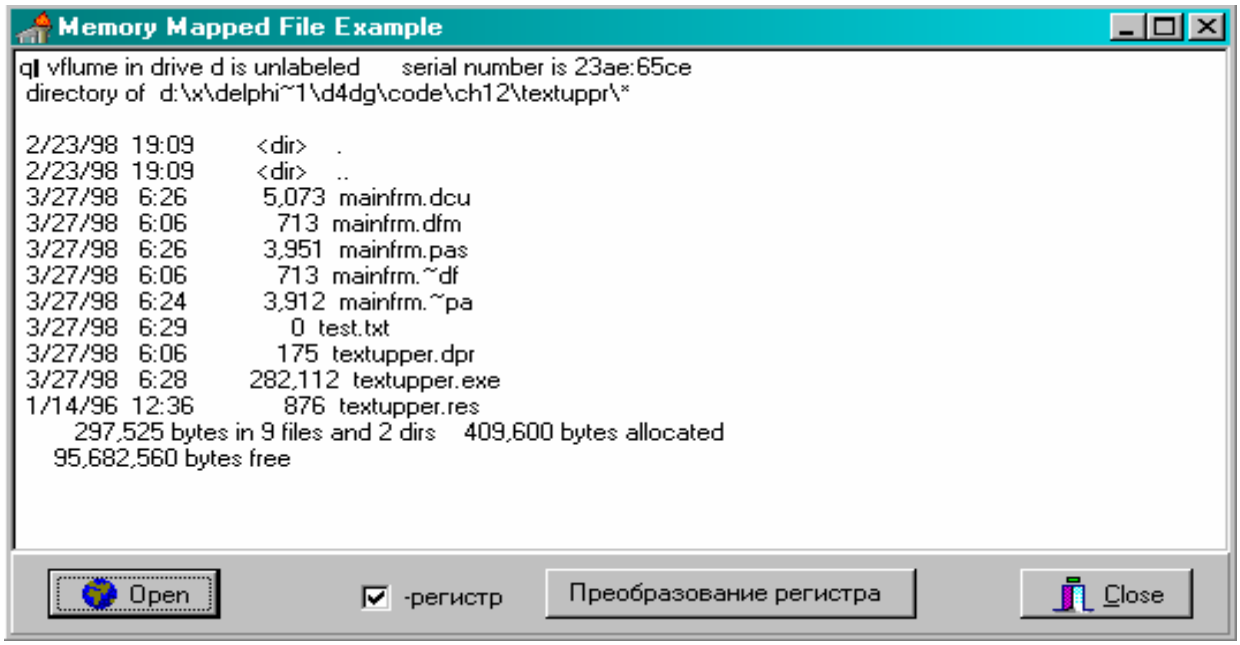

Рис.4. Внешний вид программы для отображения текстового файла в память.

Здесь в диалоге открытия вы отбираете для отображения в память произвольный текстовый файл, который выводится в компоненте TMemo. После этого вы можете преобразовать все символы файла в верхний или нижний регистр и тут же увидеть результат. Все это будет производиться с помощью функций Win API и функций работы с памятью, то есть максимально быстро. Заметим, что в нашей практике доводилось отображать бинарные файлы размером более трех мегабайт с целью проведения их предобработки-декодирования, и этот процесс на ПК 130 Mhz не занимал ощутимого времени. Это ожидаемый результат, так как ведь именно с помощью данного механизма ОС Windows загружает и выполняет EXE-и DLL-файлы. Кроме того, отображенные файлы позволяют организовать разделение их данных между различными процессами, выполняющимися одновременно на одном компьютере. Сейчас мы отметили три области применения этой методики, но рассмотрим только собственно отображение файлов, так как именно оно может заинтересовать разработчика.

Доступ к содержимому отображенных в памяти файлов

В результате отображения дисковый файл связывается с некоторой областью виртуальной памяти процесса (напомним, что в Windows 9x/NT/2000/XP каждому процессу выделяется 4 Гб виртуального адресного пространства). Для организации этой связи необходимо выполнить последовательно несколько этапов: создать или открыть файл, создать ***объект отображения файла*** (file-mapping object), создать специальное окно просмотра (file view). Далее система сама выполняет кэширование, буферизацию и загрузку данных файла в это окно просмотра, а также управляет выделением и освобождением памяти. Нам остается только организовать редактирование данных.