## ЛАБОРАТОРНАЯ РАБОТА №1

***Разработка простого приложения Delphi***

**1. Цель работы**

Ознакомление со средой визуального программирования Delphi и разработка в ней простейших приложений.

**2. Домашнее задание**

Изучить разделы конспекта лекций, описывающие среду визуального программирования и палитру компонентов Standard.

Ознакомиться с описанием и заданием на лабораторную работу. Разработать (на бумаге) внешний вид экранной формы.

**3. Основные понятия и приемы**

*3.1.Работа с формами*

Проектирование форм – ядро визуальной разработки в среде Delphi. Каждый помещаемый в форму компонент или любое задаваемое свойство сохраняется в файле, описывающем форму (DFM-файл), а также оказывает некоторое влияние на исходный текст, связанный с формой (PAS-файл).

Можно начать новый пустой проект, создав пустую форму, или начать с существующей формы (используя различные доступные шаблоны), или добавить в проект новые формы. Проект (приложение) может иметь любое число форм.

При работе с формой можно обрабатывать свойства самой формы, свойства одного из ее компонентов или нескольких компонентов одновременно. Чтобы выбрать форму или компонент, можно просто щелкнуть по нему мышью или воспользоваться Object Selector ( комбинированный список в Object Inspector), где всегда отображены имя и тип выбранного элемента. Для выбора нескольких компонентов можно нажать клавишу Shift и щелкать по компонентам левой кнопкой мыши.

SpeedMenu формы содержит ряд полезных команд. Для изменения относительного расположения компонентов одного вида можно использовать команды Bring to Front и Send To Back. Командой Revert To Inherited можно воспользоваться, чтобы в унаследованной форме установить те значения свойств выбранного компонента, которые были у них в родительской форме. При выборе сразу нескольких компонентов вы можете выровнять их или изменить их размеры.

С помощью SpeedMenu можно также открыть два диалоговых окна, в которых устанавливается порядок обхода визуальных управляющих элементов и порядок создания невизуальных управляющих элементов. Команда Add To Repository добавляет текущую форму в список форм, доступ-

ных для использования в других проектах.

Для установки положения компонента, кроме применения мыши, имеются еще два способа:

- Установка значений для свойств Top и Left.
- Использование клавиш курсора при нажатой клавише Ctrl.

Метод Ctrl+клавиша курсора особенно удобен при тонкой подстройке положения элемента. Точно так же, нажимая клавиши курсора при нажатой клавише Shift, можно подстроить размер компонента.

### *3.2. Палитра компонентов*

Чтобы добавить в текущую форму новый компонент, нужно выбрать его на одной из страниц палитры Components, щелкнув по нему мышью, а затем, чтобы разместить новый элемент, щелкнуть в форме. Причем в форме можно или буксировать мышь с нажатой левой кнопкой, чтобы установить сразу и размер, и положение компонента, или просто щелкнуть один раз, позволяя Delphi установить размер по умолчанию.

Каждая страница палитры содержит ряд компонентов, которые обозначены пиктограммами и именами, появляющимися в виде подсказки. Эти имена являются официальными названиями компонентов. В действительности это названия классов, описывающих компоненты без первой буквы T (например, если класс называется Tbutton, имя будет Button). Если необходимо поместить в форму несколько компонентов одного и того же вида, то при выборе компонента щелчком в палитре удерживайте нажатой клавишу Shift. Затем при каждом щелчке в форме Delphi будет вставлять новый компонент выбранного вида.Чтобы остановить эту операцию, просто щелкните по стандартному селектору (пиктограмма стрелки)  слева от палитры Components.

### *3.3. Object Inspector*

Object Inspector используется при проектировании формы для установки свойств компонента (или самой формы). В его окне в двух колонках изменяемого размера перечислены свойства (или события) выбранного элемента и их значения. Окно Object Selector в верхней части Object Inspector указывает текущий компонент и его тип данных, и этот селектор можно использовать для изменения текущего выбора.

В Object Inspector перечислены не все свойства компонентов, а только те, что могут быть установлены на этапе проектирования. Правая колонка Object Inspector разрешает правильное редактирование в соответствии с типом данных свойства. В зависимости от свойства можно вставить строку или число, выбрать из списка опций (на эту возможность указывает стрелка) или вызвать специальный редактор (об этом говорит овальная кнопка). Для некоторых свойств, таких как Color, разрешен и ввод значения, и выбор элемента из списка, и вызов специального редактора.

### *3.4. Написание кода*

При проектировании формы в Delphi обычно приходится писать кое-какой код для отклика на некоторые из ее событий. Когда вы нажимаете кнопку мыши в форме или на компоненте, Windows посылает вашему приложению сообщение, информируя его об этом событии. Реакция Delphi состоит в получении сообщения о событии и вызове соответствующего метода отклика на событие. Для каждого вида компонентов Delphi определяет ряд различных событий. Вы можете узнать о событиях, доступных для формы или компонента, рассматривая страницу Events окна Object Inspector в тот момент, когда выбран необходимый элемент.

Вставим в форму компонент – кнопку – и заставим его откликаться на событие, связанное с нажатием левой кнопки мыши. При щелчке по кнопке мышью в работающей программе возникает событие OnClick (По щелчку). Пока это событие никак не обрабатывается программой, и поэтому нажатие кнопки не приведет ни к каким последствиям. Чтобы заставить программу реагировать на нажатие кнопки, необходимо написать на языке Object Pascal фрагмент программы, который называется обработчиком события. Фрагмент оформляется в виде специальной подпрограммы – процедуры.

Чтобы заставить Delphi самостоятельно сделать заготовку для процедуры обработчика события OnClick, дважды щелкните мышью по вновь вставленному компонету. В ответ Delphi активизирует окно кода и вы увидите в нем такой текстовый фрагмент:

```
procedure  TForm1.Button1Click(Sender: Tobject);
begin

end;
```

Слово **procedure** извещает компилятор о начале подпрограммы – процедуры. За ним следует имя процедуры TForm1.Button1Click. Это имя – составное: оно состоит из имени класса TForm1 и собственно имени процедуры Button1Click.

Каждый компонент принадлежит к строго определенному классу, а все конкретные экземпляры компонентов, вставляемые в форму, получают имя класса с добавленным числовым индексом. По используемому в Delphi соглашению все имена классов начинаются с буквы T. Таким образом, имя TForm1 означает имя класса с добавленным числовым индексом. По используемому в Delphi соглашению все имена классов начинаются с буквы T. Таким образом, имя TForm1 означает имя класса, созданного по образцу стандартного класса Tform. Если вы посмотрите начало текста в окне кода, то увидите следующие строки:

```
type
  TForm1 = class(TForm)
    Button1: Tbutton;
    procedure  TForm1.Button1Click(Sender: Tobject);
  private
    {private declarations}
  public
    {public declaration}
  end;
var
  Form1: TForm1;
```

Строка TForm1 = **class**(TForm) определяет новый класс TForm1, который порожден от стандартного класса TForm. Стандартный класс TForm описывает пустое Windows-окно, в то время как класс TForm1 описывает окно с уже вставленным в него компонентом *кнопка*..

Каждый раз, когда вы работаете над событием, Delphi открывает редактор с исходным файлом, связанным с формой. Редактор позволяет работать над несколькими файлами исходного текста одновременно, используя образ “записной книжки с ярлычками”. Каждая страница записной книжки соответствует отдельному файлу.

### *3.5. Компиляция проекта*

Компиляция проекта происходит при запуске его на выполнение. Транслируя проект, Delphi компилирует только те файлы, которые изменялись. Но если вы выберите в меню Project команду Build All, то будет откомпилирован каждый файл, даже если он не изменялся.

Проект содержит список файлов исходного кода, которые являются частями проекта, а также соответствующих им форм (при наличии таковых). Сначала каждый файл исходного кода превращается в откомпилированный модуль Delphi – файл с тем же именем, что и у исходного файла на языке Паскаль, но с расширением DCU.

Во время компиляции и компоновки самого проекта откомпилированные модули, составляющие проект, сливаются (или связываются) друг с другом и с кодом библиотеки VCL, образуя исполнимый файл.

### *3.6. Интегрированный отладчик*

В Delphi имеется встроенный отладчик, обладающий огромным количеством возможностей. При каждом запуске из среды Delphi программа уже выполняется в отладчике. Для установки точки останова или щелкните в промежутке между левой рамкой окна редактирования и текстом, или выберите в SpeedMenu команду Toggle Breakpoint, или нажмите клавишу F5.

Когда вы разместили ряд точек останова, можете использовать команду Breakpoints меню View, чтобы открыть окно Breakpoints List. Один из пунктов в верхней части окна Breakpoints List предполагает добавление условия в точке останова так, чтобы программа выполнялась только при выполнении данного условия. Кнопка Step Over на линейке SpeedBar позволяет просмотреть выполнение операторов один за другим, а кнопка Trace Into позволяет трассировать вызываемые методы (т.е. выполнять шаг за шагом код подпрограмм).

Если программа остановлена в отладчике, вы можете проверить значение любого идентификатора (для переменных, объектов, компонентов, свойств и т.д.), который доступен в этой точке программы. Для этого существуют два способа: использовать диалоговую панель Evaluate/Modify или добавить элемент в окно Watch List. Самый простой способ открыть диалоговую панель Evaluate/Modify – выделить переменную в редакторе исходного текста, а затем выбрать команду Evaluate/Modify из SpeedMenu редактора. Вы можете устанавливать элементы наблюдения, используя команду Add Wath at Cursor в Speed Menu редактора.

### *3.6. Обращение к справочной системе Help*

Для вызова справочной системы необходимо выбрать соответствующую команду в выпадающем меню Help или отметить элемент интерфейса в исходном тексте и нажать клавишу F1. При нажатии кнопки “Разделы” в окне Help появляется диалоговое окно справочной системы Windows 95, которое позволяет вам просмотреть содержание всех файлов Help группы, отыскать ключевое слово по индексу или начать процесс поиска.

### *3.7. Меню и команды Delphi*

Чтобы выдать команду в среде Delphi, можно воспользоваться тремя основными способами:

- С помощью меню.
- С помощью полоски SpeedBar (инструментальной линейки).
- С помощью SpeedMenu (одного из локальных меню, которое активизируется при нажатии правой кнопки мыши).

**Меню File**

Команды выпадающего меню File можно использовать для работы как с проектами, так и с файлами исходного кода.

К командам, работающим с проектами, относятся New, New Application, Open, Reopen, Save Project As, Save All, Close All, Add to Project и Remove from Project. С файлами исходного кода работают команды New, New Form, New Data Module, Open, Reopen, Save As, Save, Close и Print. Основной командой является File/New, которую можно использовать для вызова экспертов, для начала работы с новым приложением, для наследования формы из уже существующей и т.д. Чтобы открыть проект или файл исход-

ного кода, с которыми вы работали последний раз, используйте команду File/Reopen.

**Меню Edit**

Стандартные возможности меню Edit применимы как к тексту, так и к компонентам формы. Можно копировать и вставлять тот или иной текст в редакторе, копировать и вставлять компоненты в одной форме или из одной формы в другую. Также можно копировать и вставлять компоненты в другое групповое окно той же формы, например, в панель или блок группы; копировать компоненты из формы в редактор, и наоборот. Delphi помещает компоненты в буфер обмена, преобразуя их в текстовое описание. Можно соответствующим образом отредактировать этот текст, а затем вставить его обратно в форму в виде нового компонента. Можно выбрать несколько компонентов и скопировать их как в другую форму, так и в текстовый редактор. Это может пригодиться, когда вам придется работать с рядом схожих компонентов. Вы сможете скопировать один компонент в редактор, размножить его нужное число раз, а затем вставить назад в форму целую группу.

**Меню Search**

Если вы выберете команду Incremental Search, то вместо того чтобы показать диалоговое окно, где вводится образец для поиска, Delphi переходит в редактор. Когда вы введете первую букву, редактор перейдет к первому слову, которое начинается с этой буквы. Продолжайте набор букв и, курсор будет последовательно переходить к словам, в начале которых будут стоять введенные символы. Эта команда очень эффективна и чрезвычайно быстра. Команда Browse Symbol вызывает Object Browser – инструмент, который можно использовать для просмотра многих деталей при исследовании откомпилированной программы.

**Меню View**

Большинство команд меню View применяются для отображения какого-либо окна среды Delphi, например Project Manager, Breakpoints List или Components List. Эти окна не связаны друг с другом. Команда Toggle Form/Unit используется для перехода от формы, над которой вы работаете к ее исходному коду, и обратно. Команда New edit window создает дубликат окна редактирования и его содержимого. В Delphi это единственный способ просмотреть два файла рядом друг с другом, поскольку редактор для показа нескольких загруженных файлов использует ярлычки. После дублирования окна редактирования могут содержать разные файлы. Последние две команды меню View можно использовать для удаления с экрана полоски SpeedBar и палитры Components, хотя при этом среда Delphi становится менее удобной для пользователя. Команда Build All заставляет Delphi откомпилировать каждый исходный файл проекта, даже если после последней трансляции он не был изменен. Для проверки написанного кода без создания программы

можно использовать команду Syntax Check. Команда Information дает некоторые подробности о последней выполненной вами трансляции. Команда Options применяется для установки опций проекта: опций компилятора и редактора связей, опций объекта приложения и т.д.

**Меню Run**

Меню Run можно было бы назвать Debug (отладка). Большинство команд в нем относится к отладке, включая саму команду Run. Программа, запускаемая внутри среды Delphi, выполняется в ее интегрированном отладчике (если не отключена соответствующая опция). Для быстрого запуска приложения используется клавиша F9. Остальные команды применяются в процессе отладки для пошагового выполнения программы, установки точек прерывания, просмотра значений переменных и объектов, и т.п.

**Меню Component**

Команды меню Component можно использовать для написания компонентов, добавления их в библиотеку, а также для конфигурирования библиотеки или палитры компонентов.

**Меню Tools**

Меню Tools содержит список нескольких внешних программ и инструментальных средств. Команда Tools позволяет сконфигурировать это выпадающее меню и добавить в него новые внешние средства. Меню Tools также включает команду для настройки репозитория и команду Options, которая конфигурирует всю среду разработки Delphi.

### *3.8. Полоска кнопок быстрого доступа SpeedBar*

Наиболее часто используемые команды Delphi имеются в инструментальной линейке SpeedBar. Изменить размеры SpeedBar можно, буксируя толстую линию между ней и палитрой Ccomponents. Другие разрешенные в SpeedBar операции добавляют, удаляют или заменяют пиктограммы с помощью команды Configure собственного локального меню SpeedBar. Эта операция вызывает инструмент SpeedBar Editor. Чтобы добавить пиктограмму в SpeedBar, необходимо найти ее в нужной категории и отбуксировать в полоску. Подобным образом можно отбуксировать пиктограмму за пределы SpeedBar или просто передвинуть ее в другое место.

### *3.9. Локальные меню SpeedMenu*

Хотя меню Delphi содержит большое количество элементов, не все команды доступны через выпадающие меню. Иногда для некоторых окон или областей окна приходится использовать SpeedMenu (локальное меню). Чтобы активизировать SpeedMenu, нужно нажать над необходимым элементом интерфейса пользователя правую кнопку мыши или клавиши Alt и F10.

### *3.10. Файлы, создаваемые системой*

Когда вы сохраняете новый проект, Delphi создает ряд файлов.

Здесь приводится список наиболее важных файлов.

- Основной файл проекта типа .DPR. Это основной модуль исходного текста проекта. Имеется только один DPR-файл для каждого проекта. Этот файл, кроме всего прочего, перечисляет имена других файлов, составляющих проект.
- Файлы формы типа .DFM. Это двоичные файлы ресурсов, содержащие определение визуальных форм. В проекте может быть много форм и каждая имеет собственный .DFM файл.
- Файл модуля Паскаля типа .PAS. Содержит код Object Pascal для соответствующей формы или для автономного модуля кода.
- Файл опций проекта типа .OPT. Файл, который содержит различные установки Delphi (текстовый файл).
- Откомпилированные файлы модуля типа .DCU. Содержат объектный код существующего .PAS-файла модуля.
- Откомпилированные программные файлы типа .EXE. Это собственно программы Windows.
- Откомпилированные файлы динамических библиотек типа .DLL. Это откомпилированные модули Windows, которые могут использоваться одновременно многими программами Windows.

### *3.11. Страницы репозитория объектов*

В Delphi есть несколько команд меню, с помощью которых вы можете создать новую форму, новое приложение, новый модуль данных, новый компонент и т.п. Эти команды находятся в меню File, а также в других выпадающих меню. Но если вместо них выдать команду File/New, Delphi откроет окно Object Repository.

Репозиторий используется для создания новых элементов любого вида: форм, приложений, модулей данных, библиотек, компонентов и т.д. Диалоговое окно Object Repository содержит несколько страниц:

- Страница New позволяет создавать новые элементы многих разных типов.
- Страница текущего проекта (в действительности на ярлычке данной страницы вы увидите имя проекта, например Project1) позволяет унаследовать форму или модуль данных от аналогичного объекта, включенного в ваш текущий проект.
- Страницы Forms, Dialogs , Data Modules позволяют создавать новые формы, диалоговые панели и модули данных, используя эксперты или существующие объекты этих типов.
- Страница Project позволяет скопировать файлы из хранящегося в репозитории проекта.

В нижней части диалогового окна Object Repository находятся три радиокнопки, с помощью которых можно указать: хотите ли вы скопировать

существующий элемент, унаследовать его или применить непосредственно, не копируя.

**Страница New**

Список элементов, которые можно создать с помощью этой страницы:

- Application создает новый пустой проект.
- Data Module создает новый пустой модуль данных.
- DLL создает новую библиотеку DLL.
- Form создает новую пустую форму.
- Text открывает в редакторе новый текстовый файл.
- Unit создает новый пустой модуль, не связанный с формой.

**Страница Forms**

Ниже приведен список необходимых для работы предопределенных форм:

- About Box – простая панель “О программе”.
- Duil List Box – форма с двумя разными списками; позволяет пользователю выбрать ряд элементов в одном списке и нажатием кнопки переместить их во второй. Кроме компонентов, эта форма содержит значительное количество не очень простого кода на языке Паскаль.

**Страница Dialogs**

Эта страница похожа на предыдущую, но содержит другие элементы.

- Dialog Expert – простой эксперт, который способен сгенерировать различные диалоговые панели с одной или несколькими страницами.
- Dialog with help – два варианта диалоговой панели с кнопкой вызова справочной информации.
- Password dialog – диалоговая панель с простым окном редактирования, которая имеет необходимые для ввода пароля опции; код отсутствует.
- Standart Dialog Box – стандартная диалоговая панель, которая доступна в двух вариантах с различным расположением кнопок.

**Страница Data Modules**

Модуль данных это особый вид формы, который никогда не появляется на экране во время выполнения и может использоваться для хранения невизуальных компонентов. Чаще всего он применяется для описания доступа к базе данных.

**Страница Projects**

Эта страница содержит схемы проектов, которые вы можете использовать в качестве стартовой площадки для создания собственного приложения.

- Application Expert – простой эксперт, в котором вы можете выбрать файловую структуру и некоторые другие элементы приложения.
- MDI Application задает ключевые элементы программы с интерфейсом

Multiple Document Interface (MDI). В этом приложении определена основная форма для окна MDI-фрейма, содержащая меню, строку состояния и инструментальную линейку. Кроме того, в нем имеется вторая форма, которую на этапе выполнения можно использовать для создания дочерних окон.

- SDI Application определяет основную форму со стандартными атрибутами современного интерфейса пользователя, включая инструментальную линейку и строку состояния, а также типичную панель About.
- Win95 LogoApplication описывает простое приложение, в котором присутствует большинство элементов, необходимых для получения логотипа Windows 95. Данная команда в основном создает SDI-приложение с компонетом RichEdit и вставляет в него код, который делает приложение совместимым с электронной почтой.

### *3.12. Эксперты Delphi*

Delphi разрешает не только копировать или использовать существующий код, но и создавать новые формы, приложения или другие файлы с кодом, применяя эксперт. Эксперт позволяет вам ввести ряд опций и с помощью некоторой внутренней схемы создает код, соответствующий вашему заказу.

**Application Expert**

Его можно активизировать из страницы Project окна Object Repository. Первая страница этого эксперта позволяет добавить в программу некоторые стандартные выпадающие меню: File, Edit, Window и Help. На второй странице эксперта вы зададите расширения тех файлов, которые должна рассматривать программа. Вам придется ввести как описание файла, например, текстовый файл (*.txt), так и его расширение txt Эти величины будут использоваться в качестве значений по умолчанию диалоговыми окнами File Open и File Save, которые Application Expert добавит в программу (если вы выбрали поддержку файлов).

Application Expert выведет на экран прекрасное визуальное средство, которым вы можете воспользоваться для построения инструментальной линейки. В нем вы выбираете одно из выпадающих меню, и появляется ряд стандартных кнопок, которые соответствуют его типичным элементам (но только, если это меню было выбрано на первой странице эксперта).

После завершения работы над инструментальной линейкой вы можете перейти на последнюю страницу эксперта. Здесь устанавливаются многие дополнительные опции, например, можно доказать поддержку интерфейса MDI, добавить строку состояния или разрешить всплывающие подсказки. Вы также можете задать имя нового приложения и указать каталог для его исходных файлов. Каталог для приложений должен уже существовать. Если вы хотите поместить файлы проекта в новый каталог, выберите

кнопку Browse и введите новый путь в появившемся диалоговом окне.

**Dialog Box Expert**

Это простой эксперт, предоставленный вместе с исходным текстом в качестве демонстрационного примера. Вы можете воспользоваться этим экспертом как инструментом для построения диалоговых панелей двух различных видов: простых и многостраничных диалоговых панелей. Если вы выберите простую диалоговую панель, эксперт перейдет к третьей странице, где вы сможете задать компоновку кнопок. Если вы выберите многостраничную панель, появится промежуточная страница, которая позволяет ввести тексты для различных ярлычков.

## 4. Порядок выполнения работы

1. Войдите в среду Delphi, дважды щелкнув мышью на пиктограмму Delphi или через меню “Пуск”.
2. Совершите экскурс в среду визуального программирования Delphi. При этом результаты своей работы не сохраняйте на диске.
3. Попытайтесь создать различные приложения с помощью Expert-ов Delphi, исследовать в окне редактирования полученный код. Приложения на диске не запоминать.
4. Выберите пункт меню File/New Application, этим вы создадите новый проект приложения.
5. Выберите команду File/Save Project As…
   - В появившемся диалоге перейдите к корневому каталогу диска C, выбрав соответствующий пункт в выпадающем списке вверху окна.
   - Откройте папку, соответствующую названию вашей группы (например, P-123). Если такой нет, то создайте её, щелкнув правой кнопкой мыши на свободном месте, выбрав из появившегося контекстного меню пункт Создать/папку и введя нужное название (после чего не забудьте её открыть).
   - Создайте (как описано в предыдущем пункте) папку "Ваша_фамилия Lab1".
   - Сохраните Unit1.pas под новым именем Main.pas, а Project1.dpr под новым именем Lab1.dpr.

6. Выберите из политры Standard визуальных компонентов и поместите в форму следующие компоненты:

   - Окно редактирования со связанной с ним меткой Operand 1. В этом окне вводится первый операнд.
   - Окно редактирования со связанной с ним меткой Operator. В этом

окне вводится операция. В программе следует предусмотреть операции +, -, / и * .

- Окно редактирования со связанной с ним меткой Operand 2. В этом окне вводится второй операнд.
- Окно редактирования со связанной с ним меткой Result. В этом окне отображается результат запрошенной вами операции.

Пример простого калькулятора приведен на рис.1.1.

1. В событиях OnClick каждой кнопки опишите соответствующие действия, например, для кнопки 

код события должен выглядеть следующим образом:

```
If Edit2.Text='0' Then ShowMessage('Знаменатель равен "0"!')
Else
    Edit3.Text:=FloatToStr(StrToFloat(Edit1.Text)/StrToFloat(Edit2.Text));
```

Обратите внимание, что в данном действии осуществляется проверка деления на ноль. Оператор ShowMessage выдает сообщение, представленное на рис. 1.2 в случае, когда Y равен "0".

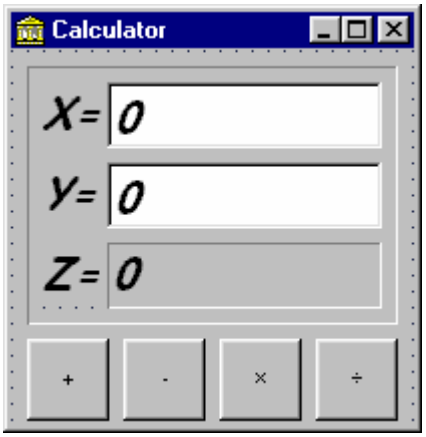

Рис. 1.1. Пример формы калькулятора

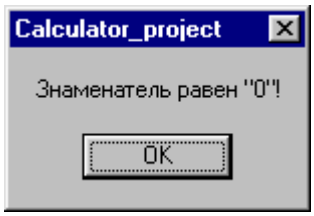

Рис. 1.2. Сообщение об ошибке

Процедуры FloatToStr и StrToFloat выполняют преобразования из числовой переменной в строковую и из строковой в числовую соответственно.

Остальные клавиши должны быть описаны соответственно.

2. Измените программу, дополнив ее возможностями очистки содержимого полей (например, кнопка Clear) и вычисления значения заданной функции, согласно варианту индивидуального задания.
3. Сохраните форму и проект.

**5. Варианты индивидуальных заданий**

1. Создайте кнопку для нахождения значений функции Z=Sin(2X+3Y).
2. Создайте кнопку для нахождения значений функции Z=Cos(4X-2Y).
3. Создайте кнопку для нахождения значений функции Z=Tan(2X+Y).
4. Создайте кнопку для нахождения значений функции Z=X! При этом в поле Y копируйте значение X.
5. Создайте кнопку для нахождения значений функции Z=Log(2X+Y+3).
6. Создайте кнопку для возведения X в целочисленную степеньY.

**6. Результаты работы**

В результате выполнения лабораторной работы студент должен продемонстрировать преподавателю готовый проект калькулятора, файл формы и исходный код модуля.

## ЛАБОРАТОРНАЯ РАБОТА № 2

***Обработка исключительных ситуаций в Delphi. Восприятие ввода от пользователя***

**1. Цель работы**

Ознакомление с классом исключительных ситуаций Delphi и создание приложений, генерирующих исключения и обрабатывающие различные фокусы ввода.

**2. Домашнее задание**

Изучить разделы 5 и 6 конспекта лекций, ознакомиться с описанием и заданием на лабораторную работу.

**3. Основные понятия и приемы**

*3.1. Обработка исключений*

На этапе выполнения Delphi порождает исключения, когда какой-либо процесс идет неправильно. Если код вашей 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.

Весь механизм строится на четырех ключевых словах:

- ✓ try – определяет начало защищенного блока кода;
- ✓ except – определяет конец защищенного блока кода и вводит операторы обработки исключений в следующем виде:

  on (тип исключения) do (оператор)

- ✓ finally – указывает необязательный блок, который используется для освобождения ресурсов, распределенных в блоке try перед обработкой исключения; этот блок завершается ключевым словом end.
- ✓ raise – оператор, используемый для порождения исключений. Большинство исключений, которые вы встретите при программировании на Delphi, будут порождаться системой, но вы также можете создать их в собственном коде, когда во время выполнения обнаружатся недопустимые или несогласованные данные. Кроме того, ключевое слово raise можно использовать внутри обработчика для повторного порождения исключе-

ния, т.е. для передачи его следующему обработчику.

Если вы хотите, чтобы при правильной обработке исключений программа продолжала выполняться, отключите опцию отладки Break on Exception в окне Environment Options.

### *3.2. Предопределенные обработчики исключительных ситуаций*

Ниже Вы найдете справочную информацию по предопределенным исключениям, необходимую для профессионального программирования в Delphi.

- **Exception** - базовый класс-предок всех обработчиков исключительных ситуаций.
- **EAbort** - "скрытое" исключение. Используйте его тогда, когда хотите 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:

  1) при попытке регистрации компоненты за пределами процедуры Register;

  2) 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 од-

ного класса к другому классу, не совместимому 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).
- **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.
- **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.
- **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.
- **EInvalidOpCode** - вызывается, когда процессор пытается выполнить недопустимые инструкции.
- **ESingleStep** - аналогично EBreakpoint, это исключение происходит

при пошаговом выполнении приложения в IDE Delphi.

- **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, имеющих 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.

### *3.3. Восприятие ввода от пользователя*

Обратим особое внимание на одно качество, характерное для многих управляющих элементов – фокус. Как определить какое окно или поле имеет фокус ввода? В каждый конкретный момент фокус имеет только одно поле. Вы можете перемещать фокус, используя клавишу Tab или щелкая мышью по другому компоненту. Каждый раз когда компонент получает или теряет фокус, к нему приходит соответствующее событие, которое указывает, что пользователь достиг (OnEnter) или покинул (OnExit) компонент.

При использовании компонента Edit для ввода чисел пользователь вместо цифры может набрать букву. Функции преобразования вернут код ошибки, что поможет определить действительно ли введено число. Когда можно выполнить такую проверку? Возможно, когда изменится значение блока редактирования, когда компонент потеряет фокус или когда пользователь щелкнет по некоторой кнопке в диалоговой панели. Можно просматривать входной поток в блоке редактирования и останавливать любой нечисловой код.

## 4. Порядок выполнения работы

1. В среде программирования Delphi создайте новый проект, выбрав пункт меню File/New Application.

2. Сохраните этот проект в папке"C:\Ваша_группа\Ваша_фамилия Lab2". (Unit1.pas под новым именем Main2.pas, а Project1.dpr под новым именем Lab2.dpr).
3. Разработайте приложение, обрабатывающее исключительную ситуацию, согласно вашему варианту индивидуального задания.
4. Открыть новое приложение.
5. Создать форму с пятью полями редактирования и пятью соответствующими надписями, которые поясняют, какой вид проверки осуществляет соответствующий компонент Edit. Форма также содержит кнопку для проверки содержимого первого поля редактирования.

Событие OnClick кнопки должно проверять целочисленность введенного в первое поле значения, например:

```
var
  Number, Code : Integer ;
begin
  if Edit1.Text <> ‘ ’ then
  begin
    val ( Edit1. Text, Number, Code) ;
     if Code <> 0 then
   begin
      Edit1. SetFocus ;
      MessageDlg (‘ Not a number in the first edit ‘ , mtError,
                        [ mbOK ] , 0 ) ;
    end ;
  end ;
end ;
```

- При выходе из компонента Edit2 (событие OnExit) осуществляется аналогичная проверка.

```
var
  Number, Code : Integer ;
begin
  if (Sender as TEdit ). Text <> ‘ ’ then
  begin
    val ((Sender as TEdit ). Text, Number, Code) ;
     if Code <> 0 then
   begin
      (Sender as TEdit ). SetFocus ;
      MessageDlg (‘ The edit field number  ‘ +
              IntToStr ((Sender as TEdit ). Tag) +
             ‘ does not have a      valid number’  , mtError, [ mbOK ] , 0 ) ;
    end ;
  end ;
```

```
end ;
```
Текст сообщения об ошибке может варьироваться.

- Третий компонент Edit выполняет аналогичную проверку при каждом изменении его содержимого (используя событие OnChange}.
- Записать код для события события OnKeyPress компонента Edit4 и проверить, не является ли введенный символ числом или клавишей Backspace (которая имеет числовое значение 8).

```
begin
  if not ( key in [ '0' . . '9' , # 8] } then
  begin
     Key  := # 0 ;
      MessageBeep ($ FFFFFFFF) ;
  end;
end;
```

- Для события OnEnter компонента Edit5 записать код, в котором необходимо преобразовать введенные символы в число с помощью функции StrToInt. Использовать исключение для обработки ошибки EConvertError.

**5. Варианты индивидуальных заданий**

1. Создать программу, позволяющую пользователю ввести два числа, которые программа разделит. Необходимо поместить на форму три объекта класса TEdit - два для операндов, один – для результата и кнопку (объект класса TButton), нажимая на которую пользователь выполняет деление. Исключить попытку деления на ноль а так же введения символов вместо цифр. Выдать сообщение о типе возникшей ошибке.
2. Создать программу, вычисляющую корни квадратного уравнения ($ax^2+bx+c=0$). Необходимо поместить на форму четыре объекта класса TEdit - три для коэффициентов квадратного уравнения, один – для результата и кнопку (объект класса TButton), нажимая на которую пользователь выполняет нахождение корней. Исключить ввод символов вместо цифр, получение отрицательного дискриминанта и ввод a = 0. Вывести при всех типах ошибок одно и то же сообщение.
3. Создать программу с "бесконечным" циклом типа while. В цикле увеличивать переменную I до значения, заданного пользователем. При достижении этого значения выходить из цикла с помощью возбуждения исключения EAbort. Выдать сообщение о выходе из цикла в блоке Except. Необходимо поместить на форму кнопку (объект класса TButton), которая запускает цикл; сообщение можно выдать с

помощью функции ShowMessage, или поместить на форму метку (объект класса TLabel), в которую помещается сообщение.

4. Создать программу, вычисляющую тангенс угла. Необходимо поместить в форму два компонента Tedit для ввода значения и результата и кнопку Tbutton для вычисления значения тангенса. Исключить ввод символов вместо цифр и получение значения тангенса угла 90 градусов. Предусмотреть возможность ввода значений в радианах.
5. Создать программу, вычисляющую логарифм числа. Для этого необходимо поместить в форму два компонента Tedit для ввода значения и результата и кнопку Tbutton для вычисления значения логарифма. Исключить ввод символов вместо цифр и получение значения логарифма 0.
6. Создать программу обработки исключения при обращении к несуществующему элементу массива. В форму поместите поля редактирования для ввода – вывода значений и номеров элементов массива и кнопку для обработки события.

## 6. Результаты работы

В результате выполнения лабораторной работы студент должен продемонстрировать преподавателю готовый проект, содержащий обработку исключительной ситуации, файл формы и исходный код модуля.

## ЛАБОРАТОРНАЯ РАБОТА № 3

***Создание и обработка меню***

**1. Цель работы**

Ознакомление с дизайнером меню Delphi и создание приложения, содержащего меню.

**2. Домашнее задание**

Изучить 7 раздел конспекта лекций, ознакомиться с описанием и заданием на лабораторную работу.

**3. Основные понятия и приемы**

### *3.1. Структура меню*

Обычно меню имеет два уровня. Строка меню, которая находится под заголовком окна, содержит имена выпадающих меню. Каждое выпадающее меню содержит несколько элементов. Однако структура меню очень гибка. Элемент меню можно поместить непосредственно в строку меню, а выпадающее меню внутрь другого выпадающего меню. Выпадающее меню внутри другого выпадающего меню (выпадающее меню второго уровня) встречается довольно часто, и для этого случая Windows предоставляет специальный визуальный значок – маленький треугольник справа от соответствующего пункта меню. Нередко вместо выпадающего меню второго уровня вы можете просто сгруппировать ряд опций в первоначальном выпадающем меню и поместить два разделителя: один до группы и один после.

### *3.2. Различные роли элементов меню*

Существует три основных типа элементов меню:

- Команды – элементы меню, которые используются для выдачи команды и выполнения действия. Визуально они никак не выделяются.
- Установщики состояния – элементы меню, которые используются для переключения опции в положения включено – выключено и изменения состояния какого – либо элемента. Если эти команды имеют два состояния, то в активном положении слева от них обычно стоит галочка. В этом случае выбор команды изменяет состояние на противоположное.
- Элементы вызова диалога – элементы меню, которые вызывают диалоговою панель. Реальное различие между этими и другими элементами меню состоит в следующем: с помощью этих элементов пользователь должен получить возможность исследовать вероятные действия соответствующей диалоговой панели. Такие команды должны иметь визуальный ключ в виде трех точек после текста.

### *3.3. Редактирование меню с помощью Menu Designer*

Система Delphi включает специальный редактор для меню Menu Designer. Чтобы вызвать этот инструмент, поместите компонент меню в форму и дважды щелкните по нему. Не волнуйтесь о точном положении данного компонента в форме, поскольку на результат это не влияет: само меню всегда помещается правильно – под заголовком формы. Menu Designer позволяет создавать меню путем простого написания текста команд, перемещать элементы или выпадающие меню с помощью буксировки и легко устанавливать свойства элементов. Для создания выпадающего меню второго уровня нужно выбрать команду Create submenu в SpeedMenu инструмента (локальном меню, которое вызывается правой кнопкой мыши).

### *3.4. Горячие клавиши меню*

Общее свойство элементов меню – наличие подчеркнутой буквы. Эту букву можно использовать для выбора меню с помощью клавиатуры. При одновременном нажатии клавиши Alt и клавиши с буквой на экране появляется соответствующее выпадающее меню. Чтобы определить подчеркнутую клавишу, просто поместите перед ней символ амперсанта (&), например, &File. Элементам меню можно назначить горячие клавиши. Для этого нужно указать значение для свойства ShortCut, выбрав одну из стандартных комбинаций.

### *3.5. Изменение элементов меню*

Для модификации элемента меню чаще всего используются три свойства. Свойство Checked используется, чтобы добавить или удалить галочку рядом с элементом меню. С помощью свойства Enabled элемент меню можно пригасить, после чего пользователь не сможет его выбрать. Свойство Caption представляет текст элемента меню. Изменяя текст элемента меню, вы указываете пользователю, что программа перешла в другое состояние.

## 4. Порядок выполнения работы

1. Войдите в среду Delphi.
2. Выберите пункт меню File/New Application, этим вы создадите новый проект приложения.
3. Выберите команду File/Save Project As…
   - ♦ В появившемся диалоге перейдите к корневому каталогу диска C, выбрав соответствующий пункт в выпадающем списке вверху окна.
   - ♦ Откройте папку, соответствующую названию вашей группы (например, P-123). Если такой нет, то создайте её, кликнув правой кнопкой мыши на свободном месте, выбрав из появившегося контекстного меню пункт Создать/папку и введя нужное название (после чего не

забудьте её открыть).

- Создайте (как описано в предыдущем пункте) папку "Ваша_фамилия Lab3".
- Сохраните Unit1.pas под новым именем Main.pas, а Project1.dpr под новым именем Lab3.dpr.
- Создайте проект согласно Вашему индивидуальному заданию. Подробное описание разработки меню приведено в индивидуальном задании 1.

## 5. Варианты индивидуальных заданий

*Вариант 1*

1. Выберите в палитре компонентов Delphi вкладку Standard (это самая первая вкладка и обычно она уже выбрана автоматически).
2. Выберите компонент MainMenu и поместите его на форму.

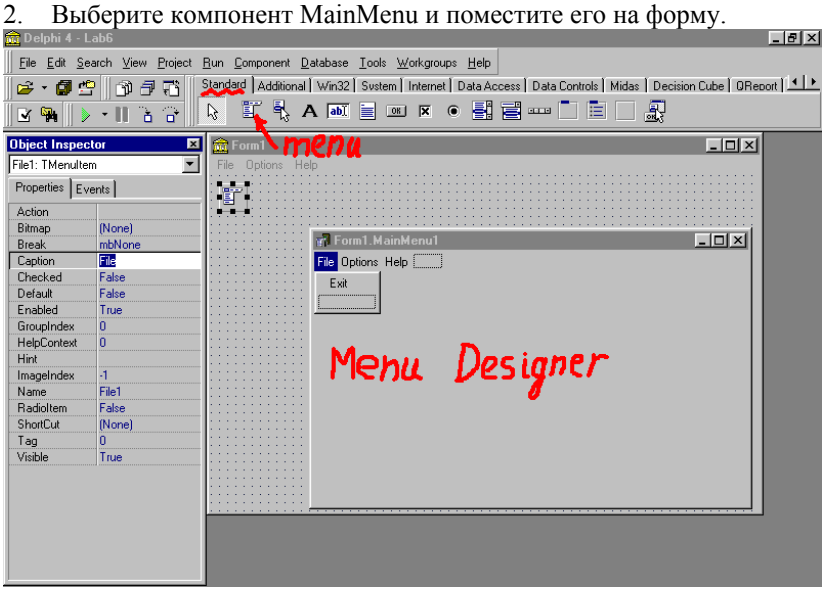


Рис. 3.1. Дизайнер меню

3. Откройте Menu Designer дважды кликнув на компоненте MainMenu в форме. Окно дизайнера меню представлено на рис. 3.1.
4. Создайте меню следующей структуры:
   - File: Exit
   - Options: Font, Color, (разделитель), Left, Center, Right, (разделитель), Height
   - Help: About

Меню должно быть похоже на представленное на рис. 3.2.
Названия пунктов меню вводятся в поле Caption, для создания разделителя в поле Caption вводится символ "-" (минус, он же дефис)


Рис. 3.2. Пример меню

5. Для Left, Center и Hight назначьте горячие клавиши (свойство ShortCut)
6. В форму поместите компонент RichEdit из палитры Win32 и две пиктограммы диалогов:
   FontDialog и ColorDialog из палитры Dialogs.
7. Для отклика на команды меню вы должны определить метод для события OnClick каждого элемента меню, как показано на рис. 3.3.

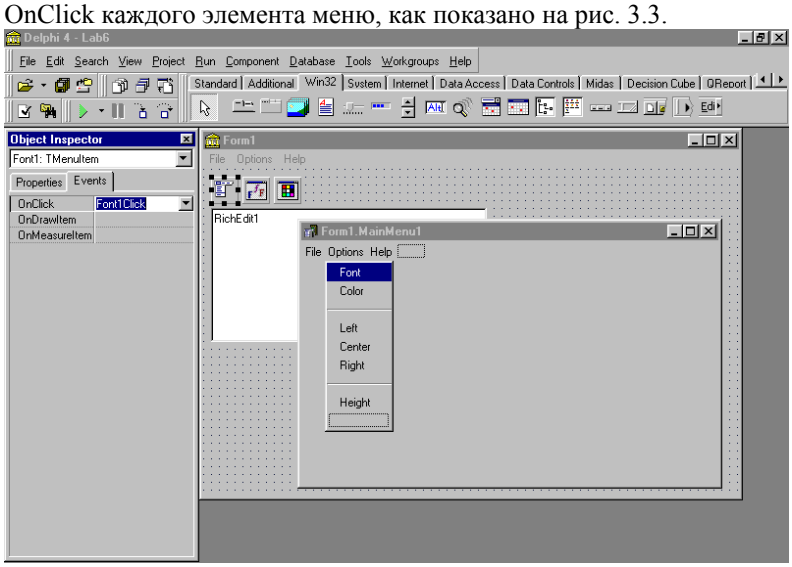


Рис. 3.3. Выбор метода OnClick

8. Код метода TForm1.Font1Click:

```
procedure TForm1.Font1Click (Sender : TObject) ;
begin
  FontDialog1.Font : = RichEdit1.Font ;
```

```
  FontDialog1.Execute;
  RichEdit1.Font : = FontDialog1.Font ;
end;
```

9. Код метода TForm1.Color1Click очень похож на приведённый выше. Напишите его сами.
10. Код метода TForm1.Left1Click:

```
procedure TForm1.Left1Click (Sender : TObject) ;
begin
  RichEdit1.Paragraph.Alignment : = taLeftJustify ;
  Left1.Checked : = True ;
  Center1.Checked : = False ;
  Right1.Checked : = False ;
end;
```

Чтобы поставить галочку в ряде выбираемых опций, установите свойство Checked элемента меню в True, как показано на рис. 3.4.

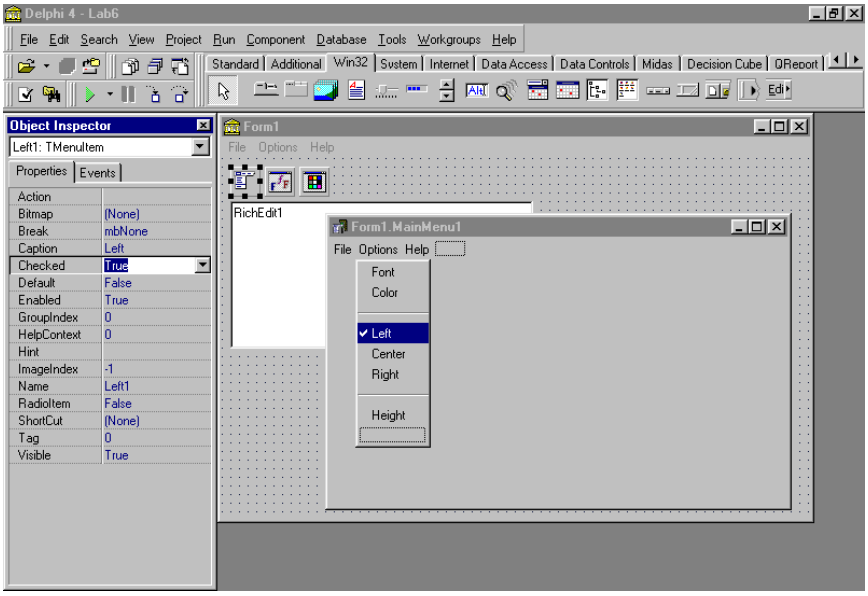


Рис. 3.4. Параметры инспектора объектов

11. Коды методов для элементов Center и Right аналогичны предыдущему.

*Вариант 2*

1. На форме расположите две панели , две кнопки и компонент RichEdit. Первая панель должна содержать два поля редактирования, а вторая два чекбокса, как показано на рис. 3.5.

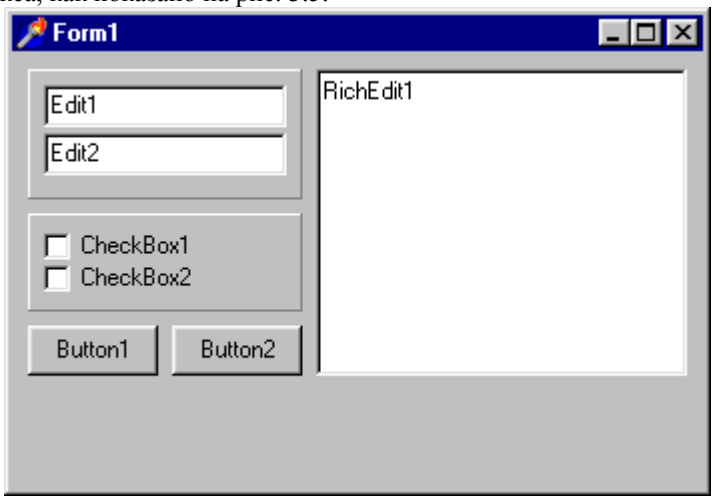


Рис. 3.5. Начальная форма проекта

2. Создайте меню:
   - File: Open, Save As
   - Buttons: Enable First (программно изменяемый на Disable First и обратно; как именно – см. ниже)
   - Views: Edit Boxes, Check Boxes
   - Pulldowns: Remove File Menu, Disable Buttons Menu

3. Поместите в форму пиктограммы диалогов OpenDialog, SaveDialog. Форма будет выглядеть примерно так, как показано на рис.3.6.

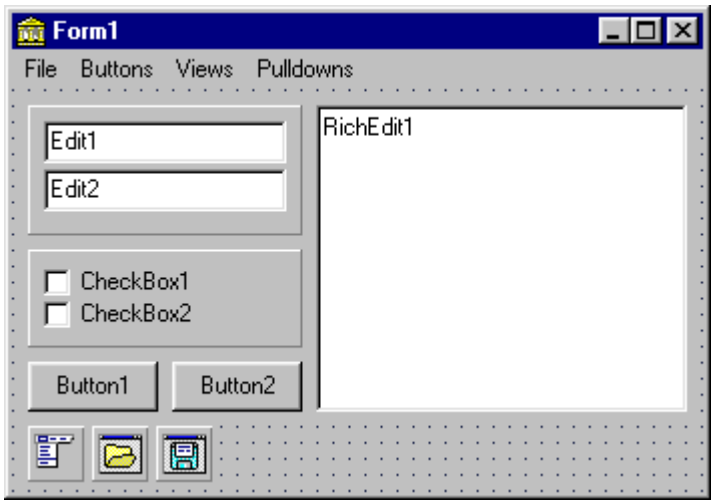


Рис. 3.6. Законченный вид формы проекта

4. Код методов, которые загружают и сохраняют файлы :

```
procedure TForm1.Open1Click (Sender : TObject) ;
begin
  if  OpenDialog1.Execute then
     RichEdit1.Lines.LoadFromFile (OpenDialog1.FileName) ;
end;

procedure TForm1.SaveAs1Click (Sender : TObject) ;
begin
  if  SaveDialog1.Execute then
     RichEdit1.Lines.SaveToFile (SaveDialog1.FileName) ;
end;
```

5. Компоненты внутри панелей в действительности не используются. Однако вам необходимо воспользоваться двумя кнопками, чтобы скрыть или отобразить каждую из двух панелей вместе с управляющими элементами, которые в них содержатся. Те же действия можно выполнить с помощью двух команд меню : View/ Edit Boxes и View/ Check Boxes. Когда вы выбираете одну из этих команд меню или нажимаете одну из кнопок, происходят три разных действия. Во-первых, отображается или скрывается панель. Во-вторых, текст кнопки изменяется с Hide на Show, и наоборот. В-третьих, рядом с соответствующим элементом меню появляет-

ся или исчезает галочка. Ниже приведен код одного из двух методов, который связан с событиями щелчка как команды меню, так и кнопки :

```
procedure TForm1.ViewEdit1Click (Sender : TObject) ;
begin
  Panel1.Visible : = not Panel1.Visible ;
   ViewEdit1.Checked : = not ViewEdit1.Checked ;
    if Panel1.Visible then
        Button1.Caption : = 'Hide' ;
    else
        Button1.Caption : = 'Show' ;
end;
```

6. Команды меню Buttons применяют другой подход. Для показа текущего состояния они используют не галочку, а изменение текста. Кроме того, они разрешают или запрещают соответствующую команду View и кнопку.

```
procedure TForm1.ButtonsFirst1Click (Sender : TObject) ;
begin
  if Buttons1.Enabled  then
  begin
     Buttons1.Enabled  : = False;
      ViewEdit1.Enabled : = False ;
       ButtonsFirst1.Caption : = 'Enable &First' ;
  end
  else
  begin
     Buttons1.Enabled  : = True
      ViewEdit1.Enabled : = True ;
       ButtonsFirst1.Caption : = 'Disable &First' ;
  end
end;
```

7. Команды меню Pulldowns должны скрывать выпадающее меню File и Duttons соответственно и показывать галочку для выбранного элемента. Запишите код для каждого элемента этого меню самостоятельно.

*Вариант 3*

1. На форме расположите компонент Image, пиктограмму диалога OpenPictureDialog

2. Создайте меню (рис. 3.7):

- File: Open, (разделитель), Exit, (разделитель) - невидимый, Most Recent - невидимый
- Options: Center, Stretch, Transparent
- About

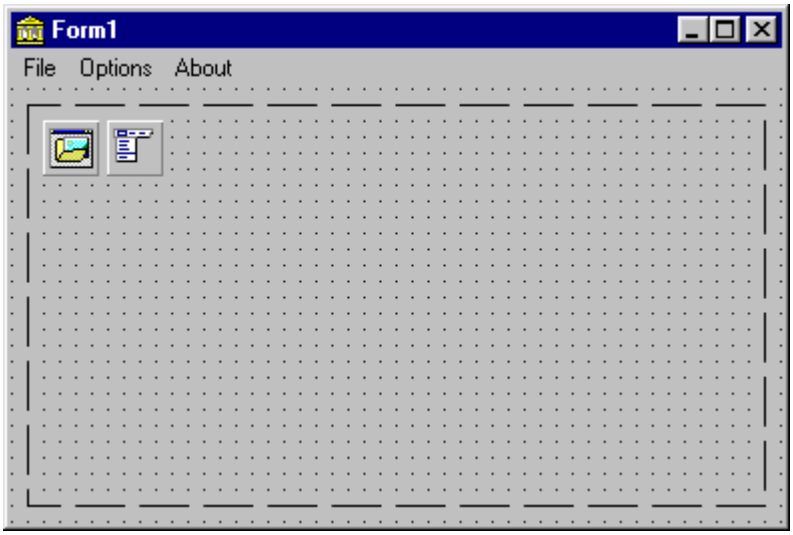


Рис. 3.7. Форма проекта варианта 3

3. Методы меню File:

```
procedure TForm1.Open1Click(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
  begin
    if Current<>'' then
    begin
      MostRecent1.Caption := Current;
      N2.Visible := true;
      MostRecent1.Visible := true;
    end;
    Current := OpenPictureDialog1.FileName;
    Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
  end;
end;
```

```
procedure TForm1.MostRecent1Click(Sender: TObject);
var
  S : string;
begin
  S := MostRecent1.Caption;
  Image1.Picture.LoadFromFile(S);
  MostRecent1.Caption := Current;
  Current := S;
end;
```

4. Метод меню TForm1.Center1Click:

```
procedure TForm1.Center1Click(Sender: TObject);
begin
  Center1.Checked := not Center1.Checked;
  Image1.Center := Center1.Checked;
end;
```

5. Остальные два метода меню Options аналогичны приведённому выше TForm1.Center1Click
6. Измените меню File и соответствующие методы для отображения имён и повторного открытия не одной, а трёх последних картинок

*Вариант 4*

1. На форме расположите компоненты Edit и Button
2. Поместите в форму компонент RadioGroup с тремя пунктами: Beep, About, Exit
6. Создайте меню:
   - Normal Part: Beep, About, Exit
   - Anomal Part

Пример формы представлен на рис. 3.8.

Рис. 3.8. Форма проекта варианта 4

4. Коды методов TForm1.Beep1Click и TForm1.Button1Click:

```
procedure TForm1.Beep1Click(Sender: TObject);
begin
  MessageBeep($FFFFFFFF);
  if (Sender as TMenuItem).Owner=AnomalPart1 then AnomalPart1.Remove(Sender as TMenuItem);
end;

procedure TForm1.About1Click(Sender: TObject);
begin
  Application.MessageBox('!!!', '...', 0);
  if (Sender as TMenuItem).Owner=AnomalPart1 then AnomalPart1.Remove(Sender as TMenuItem);
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  I : TMenuItem;
begin
  I := TMenuItem.Create(AnomalPart1);
```

```
  I.Caption := Edit1.Text;
  case RadioGroup1.ItemIndex of
    0: I.OnClick := Beep1Click;
    1: I.OnClick := About1Click;
    2: I.OnClick := Exit1Click;
  end;
  AnomalPart1.Add(I);
end;
```

5. Коды методов TForm1.About1Click аналогичен коду метода TForm1.Beep1Click
6. Добавьте в форму компонент ImageList и добавьте в него несколько иконок или небольших картинок
7. Установите свойство меню Images указывающим на ImageList1
8. Задайте свойство ImageIndex для некоторых пунктов меню.
9. Сделайте так, чтобы любой пункт из аномальной части меню случайным образом менял свойства Caption, Checked или ImageIndex какого-либо из оставшихся элементов, если таковые имеются

*Вариант 5*

1. На форме расположите компоненты Memo и Image, пиктограммы диалогов OpenDialog, SaveDialog и OpenPictureDialog.
2. Создайте меню:

- Text: Load, Save, (разделитель), Enabled – с галочкой
- Graphics: Load, (разделитель), Center – отключен, Stretch – отключен

3. Создайте всплывающее меню с единственным пунктом Clear.

Пример формы представлен на рис. 3.9.

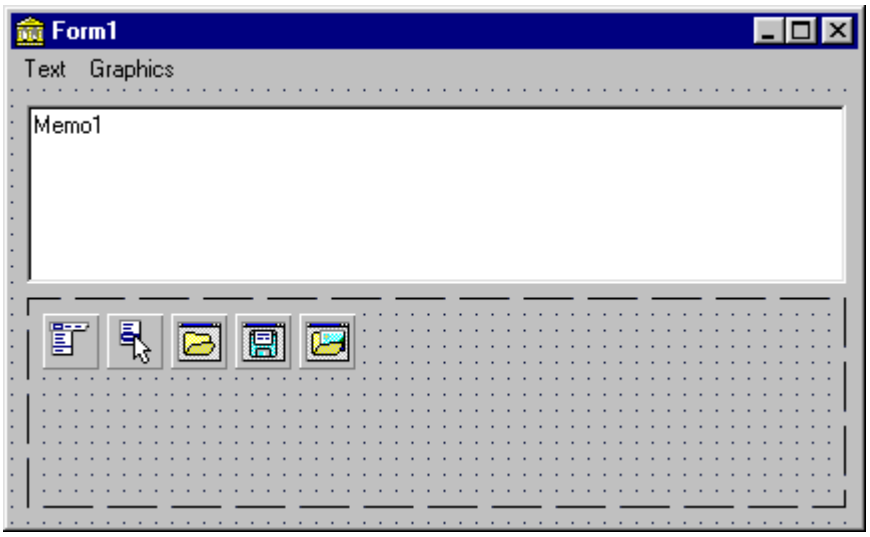

Рис. 3.9. Пример формы варианта 5

4. Коды методов:

```
procedure TForm1.Load1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;

procedure TForm1.Save1Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;

procedure TForm1.PopupMenu1Popup(Sender: TObject);
begin
  Clear1.Visible := Memo1.Lines.Count<>0;
end;

procedure TForm1.Clear1Click(Sender: TObject);
begin
  Memo1.Lines.Clear;
end;
```

```
procedure TForm1.Center1Click(Sender: TObject);
begin
  Center1.Checked := not Center1.Checked;
  Image1.Center := Center1.Checked;
end;

procedure TForm1.Load2Click(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
  begin
    Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
    Center1.Enabled := true;
    Stretch1.Enabled := true;
  end;
end;

procedure TForm1.Enabled1Click(Sender: TObject);
begin
  Enabled1.Checked := not Enabled1.Checked;
  Memo1.Enabled := Enabled1.Checked;
end;
```

5. Добавьте в главное меню пункт:

♦ Lines: Add, (разделитель) - невидимый

6. Добавьте методы:

```
procedure TForm1.Add1Click(Sender: TObject);
var
  I : TMenuItem;
begin
  I := TMenuItem.Create(Lines1);
  I.Caption := Memo1.SelText;
  I.OnClick := Put;
  Lines1.Add(I);
  N3.Visible := true;
end;

procedure TForm1.Put(Sender: TObject);
begin
  Memo1.Text := Memo1.Text + (Sender as TMenuItem).Caption;
end;
```

7. Добавьте в меню Text пункт "Alignment: Left", циклически изменяющий свойство Memo1.Alignment и своё свойство Caption для выравнивания по левому краю, по центру и по правому краю.

*Вариант 6*

1. На форме расположите три кнопки (вторая и третья - отключены), компоненты ProgressBar и Shape, пиктограммы диалога ColorDialog
2. Создайте меню:

- Progress: More, Less - отключен
- Buttons: First – с галочкой, Second, Third
- Shaping: Color

Пример формы – рис. 3.10

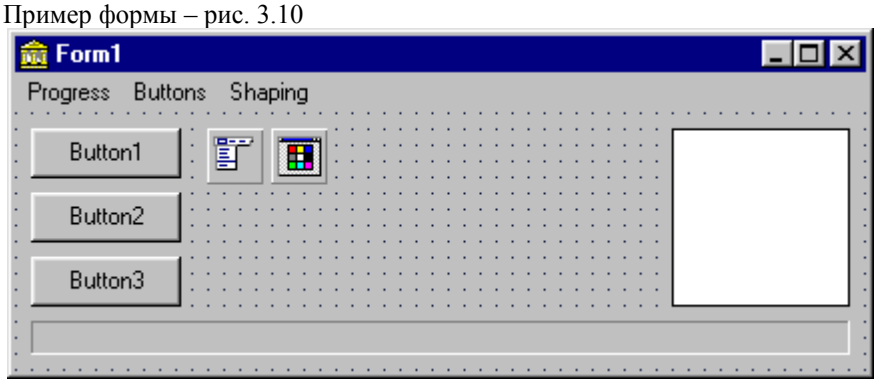


Рис. 3.10. Пример формы варианта 6

3. Коды методов:

```
procedure TForm1.More1Click(Sender: TObject);
begin
  ProgressBar1.Position := ProgressBar1.Position+10;
  Progress1.Caption := 'Progress: ' + IntToStr(ProgressBar1.Position) +
'%';
  More1.Enabled := ProgressBar1.Position<>100;
  Less1.Enabled := ProgressBar1.Position<>0;
end;

procedure TForm1.First1Click(Sender: TObject);
begin
  First1.Checked := true;
```

```
  Second1.Checked := false;
  Third1.Checked := false;
  Button1.Enabled := true;
  Button2.Enabled := false;
  Button3.Enabled := false;
end;

procedure TForm1.Color1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

4. Код метода TForm1.Less1Click аналогичен коду метода TForm1.More1Click
5. Коды методов TForm1.Second1Click и TForm1.Third1Click аналогичны коду метода TForm1.First1Click
6. Добавьте в меню Shaping пункт, циклически изменяющий свойство Shape1.Shape и отображающий эти изменения своим свойством Caption

**6. Результаты работы**

В результате выполнения лабораторной работы студент должен продемонстрировать преподавателю готовый проект, содержащий меню, файл формы и исходный код модуля.

## ЛАБОРАТОРНАЯ РАБОТА №4

***Классы и модули***

**1. Цель работы**

Объектно – ориентированное программирование в среде Delphi и разработка приложения, создающего и обрабатывающего собственный класс.

**2. Домашнее задание**

Изучить раздел 3 конспекта лекций, описывающий объектно-ориентированное программирование в Delphi.

Ознакомиться с описанием и заданием на лабораторную работу.

**3. Основные понятия и приемы**

*3.1. Классы и сокрытие информации*

Класс может содержать сколько угодно данных и любое количество методов. Однако для соблюдения всех правил объектно-ориентированного подхода данные должны быть скрыты, или инкапсулированы, внутри использующего их класса. Использование метода для получения доступа к внутреннему представлению объекта уменьшает риск возникновения ошибочных ситуаций и позволяет автору класса модифицировать внутреннее представление в будущих версиях. В Object Pascal имеются две различные конструкции, подразумевающие инкапсуляцию, защиту и доступ к переменным: классы и модули. С классами связаны некоторые специальные ключевые слова – спецификаторы доступа:

- private – элементы интерфейса класса видны только в пределах модуля, в котором определен класс. Вне этого модуля private-элементы не видны и недоступны. Если в одном модуле создано несколько классов, то все они видят private-разделы друг друга.
- Раздел public не накладывает ограничений на область видимости перечисленных в нем полей, методов и свойств. Их можно вызывать в любом другом модуле программы.
- Раздел published не ограничивает область видимости, однако в нем перечислены свойства, которые должны быть доступны не только на этапе исполнения, но и на этапе конструирования программы. Раздел published используется при разработке компонентов. Без объявления раздел считается объявленным как published. Такой умалчиваемый раздел располагается в самом начале объявления класса любой формы и продолжается до первого объявленного раздела. В раздел published среда помещает описание вставляемых в форму компонентов. Сюда не нужно помещать собственные элементы или удалять элементы, вставленные средой.

- Раздел protected доступен только методам самого класса, а также любым потомкам, независимо от того, находятся ли они в том же модуле или нет.

Порядок следования разделов может быть любой, любой раздел может быть пустым.

### *3.2. Классы и модули*

Приложения Delphi интенсивно используют модули. За каждой формой скрывается соответствующий ей модуль. Однако модули не обязаны иметь соответствующие формы.

Модуль содержит раздел interface, где объявлено все, что доступно для других модулей, и раздел implementation с реальным кодом. Наконец, модуль может иметь два необязательных раздела: initialization с некоторым кодом запуска, который выполняется при загрузке в память программы, использующей данный модуль, и finalization, который выполняется при завершении программы.

Предложение uses в начале раздела interface указывает к каким другим модулям мы должны получить доступ из раздела interface текущего модуля. Если же на другие модули необходимо сослаться из кода подпрограмм и методов, вы должны добавить новое предложение uses в начале раздела implementation.

В интерфейсе модуля можно объявить несколько различных элементов, в том числе процедуры, функции, глобальные переменные и типы данных. Также можно поместить в модуль класс. Delphi это делает автоматически при создании каждой формы. Чтобы создать новый не относящийся к форме модуль, выберите команду File/New и отметьте на странице New появившегося окна Object Repository элемент Unit.

### *3.3. Модули и область видимости*

Область видимости идентификатора (переменной, процедуры, функции или типа данных) определяет ту часть кода, в которой доступен этот идентификатор. Основное правило состоит в том, что идентификатор является значимым только внутри его области видимости.

Если вы объявили идентификатор внутри блока определения процедуры, вы не сможете использовать данную переменную вне этой процедуры. Область видимости идентификатора охватывает всю процедуру, включая вложенные блоки.

Если вы объявили идентификатор в области реализации модуля, вы не можете применить его вне модуля, но можете использовать в любом блоке и процедуре, определенных внутри модуля. Если вы объявили идентификатор в интерфейсной части модуля, его область видимости распространяется на любой другой модуль, где объявлен идентификатор. Любой идентификатор, объявленный в интерфейсе модуля, является глобальным;

все другие идентификаторы принято называть локальными.

### *3.4. Модули и программы*

Приложение Delphi создается из файлов исходного текста двух разных видов. Это один или несколько модулей и один файл программы.

Модули можно считать вторичными файлами, к которым обращается основная часть приложения – программа. На практике файл программы обычно является автоматически сгенерированным файлом с ограниченной ролью. Он нужен только для запуска программы, которая выполняет основную форму. Код файла программы, или файла проекта Delphi (DPR), можно отредактировать вручную или с помощью Project Manager и некоторых опций проекта.

### *3.5. Информация о типе на этапе выполнения*

Правило языка Object Pascal о совместимости типов для классов-потомков позволяет вам использовать класс-потомок там, где ожидается класс - предок, обратное невозможно. Теперь предположим, что класс Dog содержит функцию Eat, которая отсутствует в классе Animal.

Если переменная MyAnimal ссылается на объект типа Dog, вызов этой функции должен быть разрешен. Но если вы попытаетесь вызвать эту функцию, а переменная ссылается на объект другого класса, возникнет ошибка. Поскольку компилятор не способен определить, будет ли значение правильным на этапе выполнения, то, делая явное приведение типов, мы рискуем вызвать опасную ошибку этапа выполнения программы.

Для решения данной проблемы можно воспользоваться некоторыми подходами, основанными на системе RTTI. Каждый объект знает свой тип и своего предка и может получить эту информацию с помощью операции is. Параметрами операции is являются объект и тип:

```
if MyAnimal is Dog then …
```

Выражение is становится истинным, только если в настоящее время объект MyAnimal имеет тип Dog или тип потомка от Dog. Другими словами, это выражение приобретает значение True, если вы можете без риска присвоить объект (MyAnimal) переменной заданного типа данных (Dog). Такое прямое приведение можно выполнить так:

```
if MyAnimal is Dog then
  MyDog := Dog (MyAnimal) ;
```

То же действие можно выполнить напрямую с помощью другой операции RTTI – as. Мы можем написать так:

```
MyDog := MyAnimal as Dog ;
Text := MyDog. Eat ;
```

Если мы хотим вызвать функцию Eat, можно использовать и другую форму записи:

```
(MyAnimal as Dog) . Eat ;
```

Результатом выражения будет объект с типом данных класса Dog, поэтому вы можете применить к нему любой метод этого класса.

Приведение с операцией as отличается от традиционного приведения тем, что в случае несовместимости типа объекта с типом, к которому вы пытаетесь его привести, порождается исключение EInvalidCast.

Чтобы избежать этого исключения, используйте операцию is и в случае успеха делайте простое приведение:

```
if MyAnimal is Dog then
  (Dog (MyAnimal) ) . Eat ;
```

**4. Порядок выполнения работы**

1. В среде программирования Delphi создайте новый проект, выбрав пункт меню File/New Application.
2. Сохраните этот проект в папке "C:\Ваша_группа\Ваша_фамилия Lab4". (Unit1.pas под новым именем Main4.pas, а Project1.dpr под новым именем Lab4.dpr).
3. Открыть модуль, не связанный с формой (File/New и отметьте на странице New появившегося окна Object Repository элемент Unit), и поместить в него три класса:

- Класс Animal, который содержит в разделе public объявление конструктора Create и объявление метода-функции: Voice – звук, издаваемый животным. Тип результата возвращаемого функцией, – string. Метод Voice объявить виртуальным и абстрактным.

```
public
constructor Create;
function GetKind: string;
function Voice: string; virtual; abstract;
```

В разделе private класса определить переменную Kind: string.

- Класс Dog объявить потомком класса Animal:

```
TDog = class (TAnimal)
```

В разделе public этого класса объявить конструктор и методы Voice и Eat. Метод Eat типа string объявить виртуальным (пища животного).

```
public
constructor Create;
function Voice: string; override;
function Eat: string; virtual;
```

- Класс Cat объявить потомком класса Animal.

```
TCat = class (TAnimal)
```

Раздел public класса содержит те же определения, что и соответствующий раздел класса Dog.

```
public
constructor Create;
function Voice: string; override;
function Eat: string; virtual;
```

В реализациях конструктора каждого класса переменной Kind присваивается имя соответствующего животного, например, для класса Animal:

```
Kind := 'An Animal'.
```

В реализациях методов Voice возвращается звук, издаваемый животным, например:

```
Voice := 'Mieow'.
```

В реализациях методов Eat возвращается название пищи, которой питается соответствующее животное:

```
Eat:= 'A bone, please!';
```

4. Задать имя модуля и имя проекта, в который этот модуль будет включен.
7. Добавить в проект форму, которой присвоить имя Animals, также задать имя модулю, связанному с формой.
8. В форме расположить три кнопки опций (компонент RadioButton) с названиями Animal, Dog, Cat ; кнопкой команды (компонент Button) с названием Kind и две крупные надписи (компонент Label)(рис. 4.1). Нажатию одной из кнопок опций будет соответствовать выбор животного. При нажатии кнопки команды надписи должны отобразить звук, издаваемый животным, и его пищу.

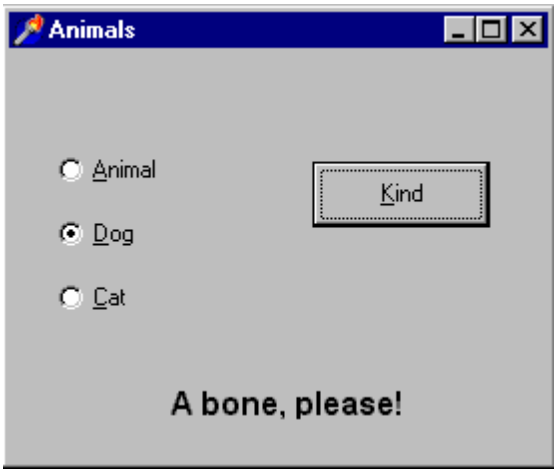

Рис. 4.1. Форма проекта

9. Определите в классе формы private-переменную MyAnimal.

```
private
MyAnimal: TAnimal;
```

Запишите код для обработчика события OnCreate формы, где создается объект типа Dog, на который ссылается переменная MyAnimal.

```
begin
MyAnimal := TDog.Create;
end;
```

10. В обработчиках события OnClick каждой кнопки опций записать код, который удаляет текущий объект и создает новый.

```
begin
  MyAnimal.Free;
  MyAnimal := TAnimal.Create;
  end;
```

11. В обработчике события OnClick кнопки команды записать код, который будет помещать в надписи звук, издаваемый животным.

```
begin
LabelVoice.Caption := MyAnimal.Voice;
  end;
```

Рис. 4.2.Звук для Cat

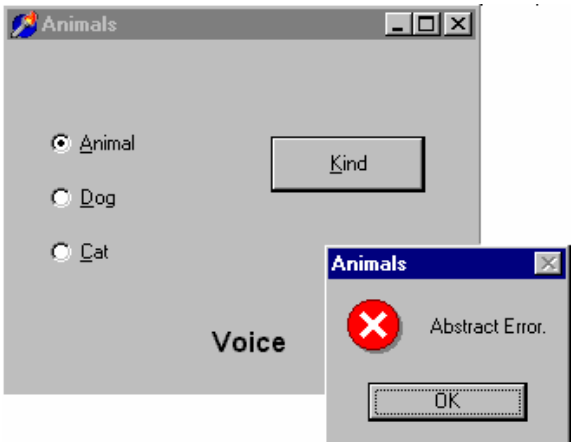

Рис. 4.3. Сообщение об ошибке

12. Если вы все сделали правильно, при запуске приложения надписи будут отображать пищу и звук для Dog и Cat (рис. 4.2) и приложение завершит работу по ошибке при выборе Animal (рис. 4.3).
13. Уберите ключевое слово abstract в объявлении метода Voice. Запустите приложение снова. Посмотрите, что изменилось в работе приложения. Объясните различия.
14. Попробуйте использовать метод Eat без приведения типов (без is).
15. Доработайте проект согласно варианту индивидуального задания.

## 5. Варианты индивидуальных заданий

**Вариант 1**

Разработайте два класса потомка от Animal, которые будут отображать особенности двух пород собак. Разработайте методы для этих классов, позволяющие получить некоторые характеристики породы (рост, длина шерсти, длина ушей и т.д.). Дополните форму компонентами, позволяющими увидеть все характеристики разработанных классов.

Примерная форма проекта представлена на рис. 4.4.

После запуска программы представляется возможность выбора между значениями: Animal, Dog, Cat. Выбор осуществляется при помощи кнопки SELECT. Далее предоставляется дополнительная возможность для выбора породы собаки.

При выделении знаком:

интересующей породы можно получить дополнительную информацию.

**Вариант 2**

Разработайте два класса потомка от Animal, которые будут отображать особенности двух новых животных Wolf (волк) и Jascal (шакал). Разработайте методы для этих классов, позволяющие получить некоторые характеристики этих видов животных (рост по холке, длина клыков, вес и т.д.). Дополните форму компонентами, позволяющими увидеть все характеристики разработанных классов.

**Вариант 3**

Разработайте два класса потомка от Animal, которые будут отображать особенности двух новых животных Fish (рыба) и Bird (птица). Разработайте методы для этих классов, позволяющие получить некоторые характеристики этих типов животных (среда обитания, покров тела, издаваемый звук и т.д.). Дополните форму компонентами, позволяющими увидеть все характеристики разработанных классов.

**Вариант 4**

Разработайте два класса потомка от Animal, которые будут отображать особенности двух видов насекомых Gnat (комар) и Fly (муха). Разработайте методы для этих классов, позволяющие получить некоторые характеристики этих насекомых (скорость передвижения, окраска, используемая пища и т.д.). Дополните форму компонентами, позволяющими увидеть все эти характеристики разработанных классов.

**Вариант 5**

Разработайте два класса потомка от Animal, которые будут отображать особенности двух видов млекопитающих Man (человек) и Monkey (обезьяна). Разработайте методы для этих классов, позволяющие получить некоторые характеристики этих существ (способ общения, покров, рост и т.д.). До-

полните форму компонентами, позволяющими увидеть все эти характеристики разработанных классов.

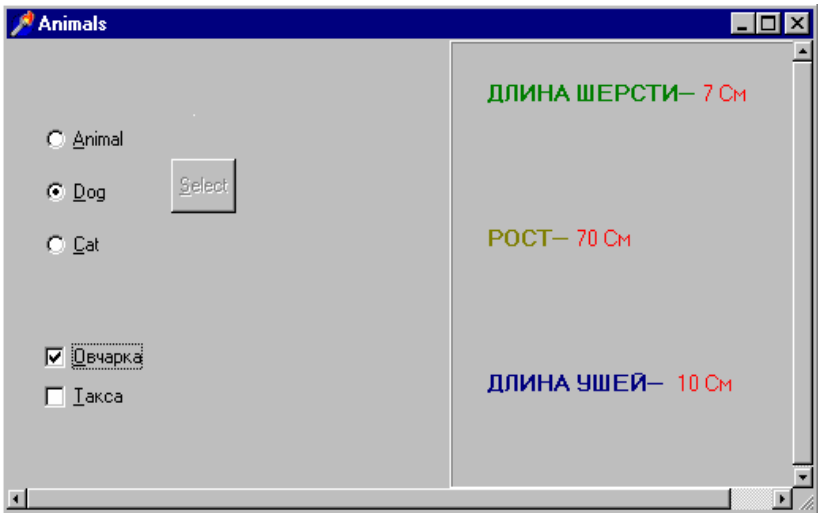


Рис. 4.4. Форма характеристик пород собак

**6. Результаты работы**

В результате выполнения лабораторной работы студент должен продемонстрировать преподавателю готовый проект, файлы форм и исходный код модулей.

# ЛАБОРАТОРНАЯ РАБОТА № 5

***Получение ввода от мыши. Рисование в форме***

**1. Цель работы**

Ознакомление с графическими средствами среды программирования Delphi и разработка в ней простейших графических программ.

**2. Домашнее задание**

Изучить 9 раздел конспекта лекций, описывающий средства черчения и рисования в форме.

Ознакомиться с описанием и заданием на лабораторную работу.

**3. Основные понятия и приемы**

Когда пользователь нажимает одну из кнопок мыши, указатель которой находится над формой (или над компонентом), Windows посылает приложению несколько сообщений. Для написания кода, откликающегося на эти сообщения, Delphi определяет несколько событий. Основных событий два: OnMouseDown, которое происходит при нажатии одной из кнопок мыши, и OnMouseUp, которое происходит при освобождении кнопки.

Еще одно важное системное сообщение связано с перемещением мыши – сообщение OnMouseMove. Событие OnClick также доступно и в форме. Его основной смысл состоит в том, что левая кнопка мыши нажимается и отпускается над одним и тем же окном или компонентом. Однако в период между этими двумя действиями курсор может переместиться за пределы окна или компонента, причем левая кнопка мыши будет удерживаться нажатой. Если вы в определенной позиции нажмете кнопку мыши, а затем переместите мышь в другое место и отпустите кнопку, то щелчок не произойдет. В этом случае окно получает только сообщение о нажатии, несколько сообщений о перемещении и сообщение об освобождении.

*3.1. События, связанные с мышью*

Метод, соответствующий событию OnMouseDown, имеет несколько параметров:

```
procedure TForm1.FormMouseDown
(Sender : TObject ;  Button : TMouseButton ;
  Shift : TShiftState ; X,Y : Integer ) ;
```

Кроме обычного параметра Sender, здесь присутствуют еще четыре:

1) Button – показывает, какая из трех кнопок мыши была нажата. Возмож-

ные значения: mbRight, mbLeft, mbCenter.

2) Shift – показывает, какие влияющие на мышь клавиши были нажаты при возникновении события. Такой клавишей может быть Alt, Ctrl или Shift, нажатая вместе с самой кнопкой мыши. Данный параметр имеет тип множества, т.к. несколько клавиш могут быть нажаты одновременно. Это означает, что при анализе условия вы должны применять не проверку на равенство, а оператор in.
3) X и Y – показывают координаты позиции мыши относительно клиентской области.

### *3.2. Рисование в форме*

Canvas (холст) – это область в форме для рисунка и многих других графических компонентов. Чтобы получить доступ к пикселям формы, используйте свойство формы Canvas и свойство Pixels для Canvas. Свойство Pixels – это двумерный массив, соответствующий цветам отдельных пикселов в Canvas. Canvas.Pixels[10,20] соответствует цвету пикселя, который находится на 10 пикселов правее и на 20 пикселов ниже точки отсчета. Обращайтесь с массивом пикселов как с любым другим свойством; чтобы изменить цвет пикселя, присвойте ему новое значение. Чтобы определить цвет пикселя, – прочитайте значение.

Этот класс создает «канву», на которой можно рисовать чертежными инструментами - пером, кистью и шрифтом. Объекты класса TCanvas автоматически создаются для всех видимых компонентов, которые должны уметь нарисовать себя. Они инкапсулируют объекты Font, Pen, Brush, а также многочисленные методы, использующие эти объекты.

Каждое свойство Canvas имеет воображаемое перо для рисования линий и контуров. Свойство Pen (перо) определяет цвет и размер линий и границ фигур. Свойствами пера являются его цвет, размер (если это сплошная линия) или стиль. Работая с пером, вы можете прочитать (но не изменить) его текущую позицию (свойство PenPos). Позиция пера определяет исходную точку следующей линии, которую программа может нарисовать с помощью метода LineTo. Для изменения позиции вы можете применить метод MoveTo канвы.

Свойство Brush (кисть) определяет цвет очерченной поверхности. Кисть используется для закрашивания замкнутых фигур. Свойствами кисти являются ее цвет, стиль и иногда растровое изображение.

Свойство Font определяет шрифт, который используется методом холста TextOut для написания текста в форме. Шрифт имеет имя, размер, стиль, цвет и т.п.

### *3.3. Черчение и рисование в системе Windows*

- Черчение – вы обращаетесь к канве и вызываете некоторые ее методы. Поскольку изображение не сохраняется, форма может частично или це-

ликом потерять свое содержимое (при закрытии окна формы другим окном или при уменьшении размера окна формы ).

- Рисование – это технология, которая позволит приложению перерисовывать всю ее поверхность при любых возможных условиях.

Для вызова перерисовки можно использовать методы Invalidate, UpDate, ReFresh и Repaint.

## 4. Порядок выполнения работы

*Задание № 1*

1. После запуска DELPHI необходимо создать форму (если она не создана автоматически) с помощью меню File\New Application. Далее необходимо создать свою папку, в которой нужно сохранить проект и модуль.
2. Поместить в форму меню Color (на странице Standart Палитры Компонентов есть объект MainMenu) с командами PenColor и BrushColor, которые будут соответственно изменять цвет пера и кисти с помощью стандартной диалоговой панели (рис. 5.1).


Рис. 5.1. Размещение меню в форме

3. В форме реализовать согласно варианту индивидуального задания рисование окружностей, эллипсов и прямоугольников различных размеров и цветов с помощью мыши, используя свойство Canvas формы.

## 5. Варианты индивидуальных заданий

**Вариант 1**

Реализовать рисование так, чтобы по нажатию мыши (щелкая левой кнопкой и удерживая ее при перемещении мыши по горизонтали и вертикали) рисовался *эллипс* нужного размера и цвета, а используя ту же технологию и удерживая клавишу SHIFT, можно было рисовать *прямоугольники* необходимого размера и цвета.

**Вариант 2**

Реализовать рисование так, чтобы по нажатию мыши (щелкая левой кнопкой и, удерживая ее при перемещении мыши по горизонтали и вертикали) рисовалась *окружность* произвольного радиуса и цвета, а используя ту же технологию и удерживая клавишу ALT, можно было рисовать *отрезки* различной длины и цвета.

**Вариант 3**

Реализовать рисование так, чтобы по нажатию мыши (щелкая левой кнопкой и удерживая ее при перемещении мыши по горизонтали и вертикали) рисовалась *окружность* произвольного радиуса, а щелкая правой кнопкой мыши и удерживая ее, можно было нарисовать *эллипс* любого размера и цвета.

**Вариант 4**

Реализовать рисование так, чтобы по нажатию мыши (щелкая левой кнопкой и удерживая ее при перемещении мыши по горизонтали и вертикали) рисовались *отрезки* произвольной длины и цвета, а используя ту же технологию и удерживая клавишу CTRL, можно было рисовать *прямоугольники* различного размера и цвета.

**Вариант 5**

Реализовать рисование так, чтобы по нажатию мыши (щелкая левой кнопкой и удерживая ее при перемещении мыши по горизонтали и вертикали) рисовалась *окружность* произвольного радиуса, а щелкая правой кнопкой мыши и удерживая ее можно было нарисовать *квадрат* любого размера и цвета.

### *5.1. Некоторые подсказки для реализации этой задачи*

Один из вариантов экрана, полученный после работы программы, представлен на рис. 5.2.

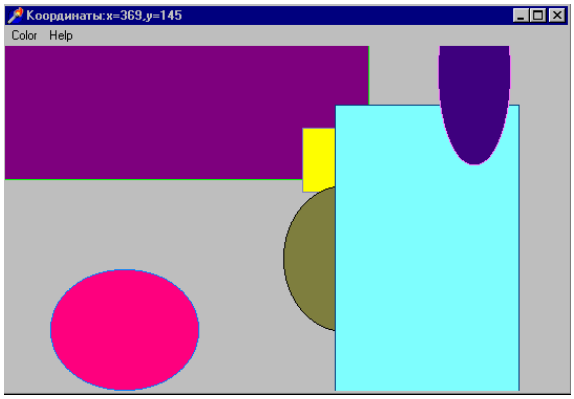


Рис. 5.2. Результат работы программы

1. Запишите следующий код для события OnMouseDown:

```
if Button = mbLeft then
  begin
    Center.X := X;
    Center.Y := Y;

 if ssShift in Shift then
      Circle := False
    else
      Circle := True;
  end;
```

Поле формы Circle типа Boolean определяет вид фигуры. Значения координат центра фигуры записываются в поля формы Center типа TPoint. Это будет выглядеть так:

```
var
  Form1: TForm1;
  Center, Radius: TPoint;
  Circle: Boolean;
```

2. Запишите следующий код для события OnMouseUp :

```
Radius.X := abs(Center.X-X);
Radius.Y := abs (center.Y-Y);
  if Circle then
     Canvas.Ellipse(Center.X-Radius.X,  Center.Y-Radius.Y,  Center.X+Radius.X, Center.Y+Radius.Y)
  else
     Canvas.Rectangle(Center.X-Radius.X,  Center.Y-Radius.Y,  Center.X+Radius.X, Center.Y+Radius.Y);
```

3. Запишите следующий код для события OnMouseMove:

```
Caption: =Format ('Координаты: x=%d, y=%d ' , [X, Y]);
```

Запустите приложение. Если все сделано правильно, то вы будете наблюдать изменение координат в заголовке формы при продвижении мыши (рис. 5.3); сможете рисовать окружности и эллипсы нужного размера (щелкая кнопкой и удерживая ее при перемещении мыши по горизонтали и вертикали); сможете рисовать прямоугольник нужного размера, используя ту же технологию и удерживая клавишу Shift.

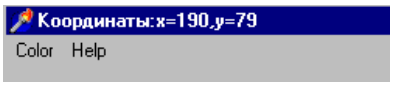


Рис. 5.3. Вывод координат в заголовке

*Задание № 2*

Начиная с оператора if, код для события OnMouseUp перенести в код для события OnPaint.

1. В код для события OnMouseUp вставить в конце вызов метода Invalidate ( который вызывает косвенно метод FormPaint, связанный с событием OnPaint). Это будет выглядеть так:

```
begin
Radius.X:=abs(Center.X -X);
Radius.Y:=abs(Center.Y-Y);
 Invalidate;
end;
```

2. Запустить приложение. При правильном выполнении всех инструкций, в форме будет рисоваться только одна фигура (рис. 5.4).

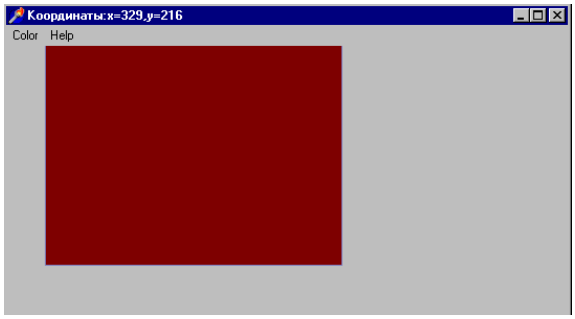


Рис. 5.4 Нарисована только одна фигура

3. Выполните первое задание для компонента PaintBox.

На странице System Палитры Компонентов есть объект TPaintBox, который можно использовать для построения приложений типа графического редактора или, например, в качестве места построения графиков. Никаких ключевых свойств, кроме Canvas, TPaintBox не имеет, собственно, этот объект является просто канвой для рисования. Важно, что координаты указателя мыши, передаваемые в обработчики соответствующих событий (OnMouseMove и др.), являются относительными, т.е. это смещение мыши относительно левого верхнего угла объекта TPaintBox, а не относительно левого верхнего угла формы.

4. Изучите возможности компонента Shape.

На странице Additionally Палитры Компонентов есть объект TShape.

TShape - простейшие графические объекты на форме типа круг, квадрат и т.п. Вид объекта указывается в свойстве Shape. Свойство Pen определяет цвет и вид границы объекта. Brush задает цвет и вид заполнения объекта. Эти свойства можно менять как во время дизайна, так и во время выполнения программы.

*Задание № 3*

Постройте график функции, значения которой вы рассчитывали в лабораторной работе № 1.

**6. Результаты работы**

В результате выполнения лабораторной работы студент должен продемонстрировать преподавателю готовые проекты графических программ, файлы форм и исходный код модулей.

## СПИСОК ЛИТЕРАТУРЫ


1. Фаронов В.В. Delphi 3. Учебный курс. М.: "Нолидж", 1998. 400 с.
2. Культин Н. Программирование на Object Pascal Delphi 5. СПб.: BHV - Санкт-Петербург, 2000. 464 с.
3. Эндрю Возневич. Delphi. Освой самостоятельно: Пер. с англ. М.: Восточная Книжная Компания, 1996. 736 с.
4. Турбо Паскаль 7.0. Киев: Издательская группа BHV, 1998. 448 с.
5. Шилдт Г. Самоучитель C++, 3-е издание: Пер. с англ. СПб.: BHV - Санкт-Петербург, 1998. 688 с.
6. Сергиевский М.В., Шалашов А.В. Турбо Паскаль 7.0: Язык, среда программирования. М.: Машиностроение, 1994. 254 с.
7. Канту М. Delphi 2 для Windows 95/NT. Полный курс. В 2 т. Т. 1: Пер. с англ. М.: Малип, 1997. 400 с.

## ОГЛАВЛЕНИЕ