МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ

Г. Г. Швачич, О. В. Овсянніков, В. В. Кузьменко, Н. І. Нєчаєва

ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Затверджено на засіданні Вченої ради академії
як конспект лекцій

Дніпропетровськ НМетАУ 2007

УДК 004 (075.8)
Швачич Г.Г., Овсянніков О.В., Кузьменко В.В., Нєчаєва Н.І.. Прикладне програмне забезпечення: Конспект лекцій. – Дніпропетровськ: НМетАУ, 2007. – 49 с.

Викладені основи об’єктно-орієнтованного програмування в інтегрованому середовищі розробки прикладного програмного забезпечення Delphi.

Призначений для студентів спеціальності 6.020100 – документознавство та інформаційна діяльність.

Іл. 17. Бібліогр.: 5 найм.

Відповідальний за випуск Г.Г. Швачич, канд. техн. наук, проф.

Рецензенти: Б. І. Мороз, д-р техн. наук, проф. (Академія таможенної Служби України)

Т. І. Пашова, канд. техн. наук, доц. (Дніпропетровський державний аграрний університет)

ТЕМА 1 ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ Delphi

Основой *Delphi* является графическая среда разработки приложений, называемая интегрированной средой разработки *(Integrated Development Environment, IDE)*. Основой создаваемого в среде *Delphi* приложения всегда является форма (*Form*). В форме (рис.1) можно размещать различные компоненты. Например, поля ввода, кнопки, таблицы, меню, панели и другие. Программный код таких компонентов автоматически генерируется *Delphi* при их установке в форму. Для создания многих приложений, порой бывает достаточно разместить в форме стандартные компоненты, так как их число в *Delphi* очень велико.

1.1 Главные составные части среды программирования

Составляющими среды программирования Delphi (рис.1) являются:

- Дизайнер Форм (*Form Designer*).
- Окно Редактора Исходного Текста (*Editor Window*).
- Палитра Компонентов (*Component Palette*).
- Инспектор Объектов (*Object Inspector*).
- Меню (*Menu System*).
- Панель инструментов (*SpeedBar*).

Имеются, конечно, и другие составляющие *Delphi*, такие как: интуитивный помощник написания кода, менеджер проекта и многие другие, используемые для точной настройки программы и среды программирования. Программирование в среде *Delphi* предполагает частое переключение между «**Дизайнером Форм**» и «**Окном Редактора Исходного Текста**» (которое для краткости называют Редактор).

«**Дизайнер Форм**» среды *Delphi* интуитивно понятен и прост в использовании. Он первоначально состоит из одного пустого окна, которое заполняется нужными объектами, выбранными в «**Палитре Компонентов**».

«**Палитра Компонентов**» позволяет выбрать нужные объекты для размещения их в «**Дизайнере Форм**». Для использования компонентов необходимо с помощью указателя мыши выбрать один из объектов, нажав и отпустив левую клавишу мыши, затем переместить курсор в рабочее поле «**Дизайнера Форм**».

Среда программирования Delphi

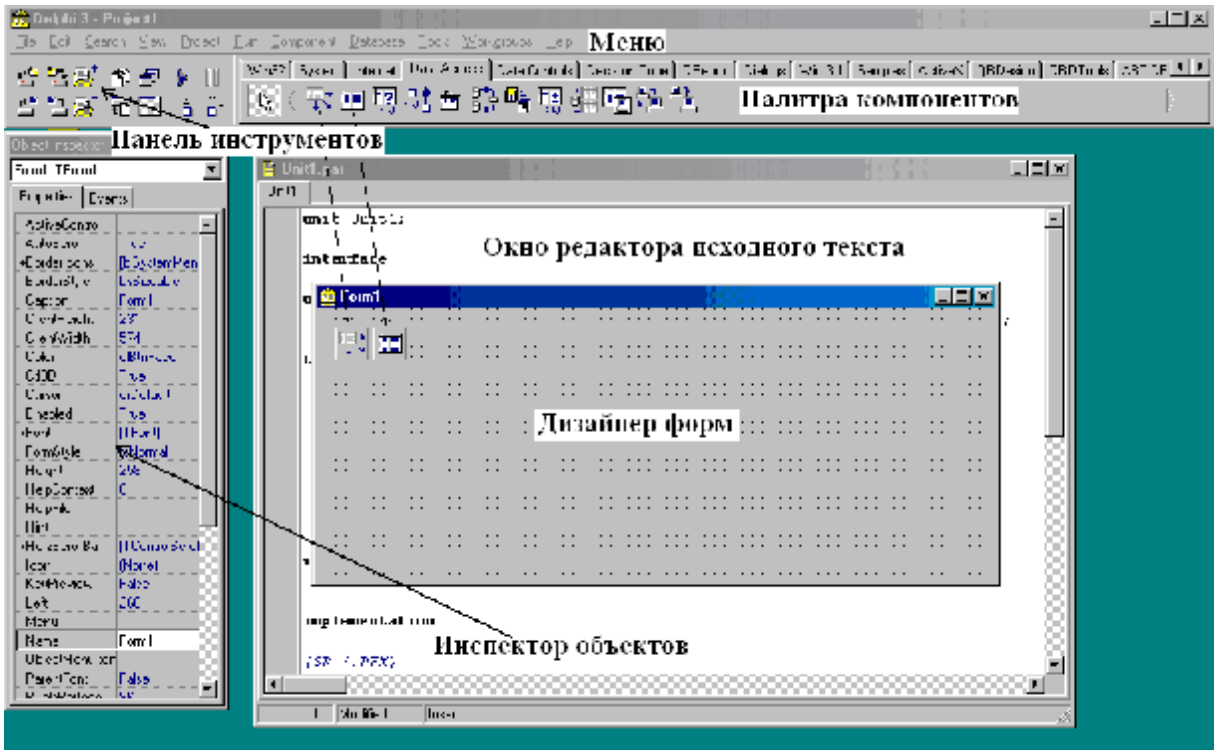

Рис.1

При установке компонента в форму ему присваивается собственное имя, включающее порядковый номер. Например, при размещении в форме двух компонентов *Edit* им будут присвоены имена *Edit1* и *Edit2* соответственно. Имя компонента является его идентификатором, поэтому двум компонентам, установленным в форму, нельзя присваивать одинаковые имена. Имена компонентов, как и другие их свойства, в дальнейшем можно изменять. Любой компонент, установленный в форму, является **Экземпляром** своего *Класса*. Например, поле ввода *Edit1* является экземпляром класса *TEdit*.

Компоненты, представленные в «**Палитре Компонентов**» среды *Delphi* также являются *Экземплярами* своего *Класса*, поэтому следует различать название *Класса* компонент и употребление *имени* компонента. Когда говориться о компоненте, как о представителе *Класса*, то употребляется слово *TComponent***,** например: *TEdit*. Когда компонент называется по имени, представленному в «**Палитре Компонентов**», то употребляется слово *Component***,** например: *Edit*.

«**Палитра Компонентов**» использует постраничную группировку объектов. В верхней части Палитры находится набор страниц – *Standard, Additional, Dialogs и др*.

Слева от «**Дизайнера Форм**» размещен «**Инспектор Объектов**». Информация в «**Инспекторе Объектов**» изменяется в зависимости от объекта, выбранного в форме. Необходимо отметить, что каждый компонент является настоящим объектом, которым можно управлять при помощи «**Инспектора Объектов**».

«**Инспектор Объектов**» состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента. Первая страница – это список свойств, вторая – список событий. Если необходимо изменить какое-либо свойство объекта, то обычно это выполняется в Инспекторе Объектов. К примеру, можно изменить имя, название и размер компонента *Panel*, изменяя свойства *Name, Caption, Left, Top, Height, и Width* в окне «**Редактора свойств**» «**Инспектора объектов**».

Для переключения между страницами свойств и событий используются закладки в верхней части «**Инспектора Объектов**». Страница событий связана с «**Редактором**». Если дважды щелкнуть мышью по правой стороне какого-нибудь пункта, то соответствующая данному событию конструкция программного кода автоматически запишется в «**Редактор**». При этом «**Редактор**» получит фокус (окно «**Редактора**» станет активным и появится на переднем плане экрана) и сразу появится возможность добавить код обработчика данного события.

Меню в среде *Delphi* предоставляет быстрый и гибкий интерфейс. Это обусловлено тем, что помимо команд системного меню управление может осуществляться при помощи «горячих клавиш». Наиболее часто используемыми в работе командами являются команды меню:

- File (New, New Application, New Form, Open, Save As, Save Project As, Save All).
- Edit (Cut, Copy, Paste, Delete).
- View (Project Manager, Project Source).
- Project (Add to Project, Options).
- Run**,** Components и Help.

Например: команда *New* предоставляет доступ практически ко всем компонентам будущего проекта и позволяет автоматизировать его разработку.

«**Панель инструментов**» *SpeedBar*, обеспечивающая быстрый дос-

туп, находится непосредственно под строкой первых пунктов системного меню, слева от «**Палитры Компонентов**».

Среда *Delphi* имеет собственные средства оформления разрабатываемых приложений. К ним относится Редактор изображений, который позволяет создавать и редактировать иконки, пиктограммы клавиш и файлы ресурсов проекта. Доступ к Редактору изображений осуществляется из меню *Tools* / *Image Editor*.

1.2 Стандартные компоненты

Набор и порядок компонентов на каждой странице в среде **Delphi** являются конфигурируемым. На первой странице «**Палитры Компонентов**» размещены 14 объектов, наиболее важных для использования. К ним относятся: *меню* и *контекстное меню*, *надписи* и *поля ввода*, *клавиши*, *опции*, *панели* и т.д.

- *TMainMenu* – позволяет поместить главное меню в программу. При этом объект *MainMenu* в форме представляется, как просто иконка. Иконки данного типа называют «*невидимыми компонентом*», поскольку они невидимы во время выполнения программы. Создание меню включает три шага: 1 – помещение *MainMenu* в форму, 2 – вызов «**Дизайнера**» Меню через свойство *Items* в Инспекторе Объектов, 3 – определение пунктов меню в Дизайнере Меню.
- *TPopupMenu* – позволяет создавать всплывающие меню. Данный компонент также является невидимым. Работа с объектом *PopupMenu* подобна работе с объектом *MainMenu*.
- *TLabel* – служит для отображения надписей в форме программы. Для надписей можно изменить цвет рамки, расположение надписи внутри рамки.
- *TEdit* – стандартный управляющий элемент *Windows* для ввода текста. Он может быть использован для отображения короткого фрагмента текста и позволяет вводить и редактировать текст во время выполнения программы.
- *TMemo* - другая форма *Edit*. Предусматривает работу с большими текстами. *Memo* может переносить слова, сохранять в буфере обмена фрагменты текста и восстанавливать их, и обладает другими основными функциями редактора текстов.

- *TButton* - это простая кнопка, которая позволяет выполнить какие-либо действия при нажатии на нее во время выполнения программы. Если поместить *Button* в форму, то двойным щелчком мыши по ней можно создать заготовку обработчика события «нажатие кнопки»:

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

Далее необходимо заполнить заготовку программным кодом. Между ключевыми словами **begin** и **end**, помещается код, который вводится вручную:

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
MessageDlg('Вы хотите изучить Delphi ?', MtConfirmation ,mbYesNoCancel ,0);
end;
```

- *TCheckBox* – **(**опция**)** отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что выбрано условие.
- *TRadioButton* **–** позволяет выбрать одну опцию из нескольких.
- *TListBox* **–** используется для показа прокручиваемого списка.
- *TComboBox* **–** поле со списком.
- *TScrollbar* **–** полоса прокрутки**,** которая применяется для просмотра большого объема информации и визуального управления характеристиками объектов.
- *TGroupBox* **–** группирующий элемент.
- *TRadioGroup* представляет собой комбинированный группирующий элемент содержащий список опций *TRadioButton***.** Количество и названия опций определяется в свойстве *Items***.**
- *TPanel* – управляющий (группирующий) элемент, похожий на *GroupBox*, применяется в декоративных целях.

1.3 Сохранение файлов проекта

Проект любой разрабатываемой программы является ее основой, а исполняемый файл (приложение) – результатом разработки. Как правило, для проектирования реальных приложений требуется значительное количество времени, что приводит к необходимости хранения файлов проекта. В проект программы можно вносить изменения в любое время, также к разрабатываемому проекту можно подключать составные части ранее разработанных проектов, что существенно сокращает время разработки. Поэтому, знание структуры проекта в целом, и его составных частей (файлов) является обязательным для разработчика программного продукта.

Папки, содержащие файлы проектов можно перемещать с диска на диск, переносить на другие компьютеры и переименовывать, но файлы, относящиеся непосредственно к проекту переименовывать нельзя. Следует отметить, что проекты приложений, разработанные в ранних версиях среды *Delphi* совместимы с поздними версиями. Однако прямой обратной совместимостью версии *Delphi* не обладают. Также, обратите внимание на то, что если в проекте использованы компоненты третьих лиц, то при перемещении на другой компьютер, соответствующие компоненты также должны быть перемещены и инсталлированы в среде *Delphi*.

Разработку любого нового проекта рекомендуется начинать с создания новой папки, которая может содержать вложенные папки для хранения дополнительной информации, отдельных проектов, входящих в комплексный проект или учебное задание.

После создания структуры папок для хранения файлов проекта необходимо выбрать команду меню *File / Save Project As*. Сохранить нужно два файла. Первый – модуль (*unit*), содержащий программный код, второй – главный файл проекта, который идентифицирует программу. Отметим, что при первом сохранении нового проекта, файлам проекта рекомендуется присваивать уникальные имена.

Предположим, что мы начали работу над новым проектом, который назовем *Labwork_1***.** Первым сохраняется файл модуля, которому присвоим имя *Lab_1.PAS***.** Вторым сохраняется главный файл проекта с именем *Labwork_1.DPR*. Сохранять файл проекта и файл модуля с одинаковыми име-

нами нельзя. Сохранять указанные файлы необходимо до компиляции программы. Это обусловлено тем, что среда *Delphi* будет знать, где в дальнейшем размещать другие файлы проекта, включая скомпилированную программу. В процессе сохранения к указанным файлам *Delphi* добавит следующие файлы: *Labwork_1.DOF, Labwork_1.RES и Lab_1.DFM.* Для создания исполняемого файла необходимо скомпилировать проект. Компиляция выполняется командой системного меню *RUN* / RUN. После выполнения этого действия в рабочей папке появятся файлы *Labwork.exe и Lab_1.DCU.*

При создании нового проекта среда *Delphi* сформирует файлы:

- *Labwork_1.DPR* – файл проекта. Он содержит код главной программы, написанной на языке *Object Pascal*. В файле проекта содержатся ссылки на все формы проекта и относящиеся к ним модули. В нем также содержится код инициализации приложения.
- *Lab_1.DFM* – файл формы, для которого декларируется тип, который определяет форму как Класс. Класс – это объектный тип. Объявление нового класса всегда содержится в отдельном модуле. В нашем случае это *Lab_1.PAS*. Каждая форма является компонентом, следовательно, и графическим объектом. Все свойства соответствующей формы хранятся в двоичном файле *Lab_1.DFM*.
- *Lab_1.PAS* – Pascal файл. Стандартный идентификатор класса формы. Этот файл содержит весь программный код, относящийся к данному модулю.
- *Labwork_1.RES* – файл ресурсов приложения. Представляет собой двоичный файл, содержащий пиктограммы, графические изображения, курсоры и строки.
- *Labwork_1.DOF* – текстовый файл, который содержит опции проекта такие как: настройки компилятора и компоновщика, имена служебных каталогов и условные директивы.
- *Lab_1.DCU* – двоичный, скомпилированный файл PAS файл.
- *Labwork_1.EXE* – исполняемый файл (Приложение). В данном случае *Labwork_1.EXE* – готовая программа, которая может функционировать под управлением операционной системы Windows.

Помимо указанных файлов *Delphi* может создавать и другие файлы. Это файлы временного хранения, имеющие расширение *~Pa, ~Df.*

ТЕМА 2 ОБЗОР ПАЛИТРЫ КОМПОНЕНТОВ

2.1 Компоненты страницы Additional

На странице **Additional** (рис.2) представлены компоненты, позволяющие создать пользовательский интерфейс программы. Ряд компонентов, содержащихся на данной странице, подобны компонентам страницы **Standard**, но обладают большими возможностями.

Компоненты, расположенные на странице Additional

Рис.2

Данная страница содержит 13 компонентов, имеющих следующее назначение:

- **TBitBtn –** кнопка похожая на **TButton**, но обладающая расширенными свойствами. На ней можно помещать картинку (**glyph**). **TBitBtn** имеет несколько предопределенных типов (**bkClose**, **bkOK** и др.), при выборе которых кнопка принимает соответствующий вид. Кроме того, нажатие кнопки в модальном окне (**MdForm.ShowModal**) приводит к закрытию окна с соответствующим модальным результатом (**MdForm.ModalResult**).
- **TSpeedButton –** кнопка для создания панели быстрого доступа к командам **SpeedBar**. Эта кнопка обладает рядом уникальных свойств, таких как *Слияние* и *Залипание*. При помощи *нескольких кнопок, объединенных в группу* легко создать переключатель или панель переключателей как в **Word** и **Excel**. Обычно на данной кнопке размещается только картинка (**glyph**) или символ.
- **TMaskEdit –** налог **TEdit**, обладающий возможностью форматированного ввода вывода. Формат определяется в свойстве **EditMask**. В редакторе свойств **EditMask** есть заготовки некоторых форматов: даты, валюты и т.д.

- **TStringGrid –**служит для представления текстовых данных в виде таблицы. Ввод данных в таблицу выполняется аналогично вводу данных в приложении **Ms Excel**. Программный доступ к каждому элементу таблицы осуществляется посредством свойства **Cell**. Программный код записи значений в ячейки **StringGrid** приведен примере 1.

Пример 1

```
Procedure TForm1.Button1Click(Sender: TObject);
begin
  StringGrid.Cells[0,0] := 'Индекс';
  StringGrid.Cells[1,1] := FloatToStr(Sqr(Sin(0.55))+Cos(Sqr(0.707)));
end;
```

- **TDrawGrid** –служит для представления данных любого типа в виде таблицы. Доступ к каждому элементу таблицы происходит через свойство **CellRect**.
- **TImage –** отображает картинку. Воспринимает файловые форматы **BMP**, **ICO**, **WMF**. Изображение может загружаться в **TImage** во время дизайна приложения и в период его выполнения. Если картинку подключить во время дизайна программы, то она компилируется в приложении (EXE файле).
- **TShape -** служит для создания и отображения простейших графических объектов на форме: окружность, квадрат и т.п.
- **TBevel** – элемент для рельефного оформления интерфейса приложения. Данный элемент не имеет обработчиков событий. Основными свойствами управления компонентом являются **Style** и **Shape**.
- **TScrollBox** – позволяет создать в форме прокручиваемую область с размерами большими, размеров экрана. На этой области можно разместить любые элементы управления.
- **TCheckListBox** – окно списка опций с линейкой прокрутки. В отличие от **TListBox** позволяет отмечать пункты исполняемого списка.
- **TSpliter** – перемещаемый распределитель, подобный распределителю положения окон **Проводника Windows**.
- **TStaticText** – компонент, подобный Label, но обладающий большими возможностями.

- **TChart** – однофункциональный компонент позволяющий отображать данные в виде двумерных **2D** и объемных **3D** графиков и диаграмм в режиме дизайна и выполнения программы. В процессе дизайна приложения доступ к настройке компонента, после установки последнего в форму, выполняется двойным щелчком мышью на рабочем поле компонента.

2.2 Компоненты страницы Win32

Компоненты, расположенные на странице Win32 (рис.3) предназначены для оформления приложений в стандарте и стиле **Windows9x**.

Компоненты, расположенные на странице Win32

Рис.3

Всего на странице находятся 16 компонентов, назначение которых приводится ниже:

- **TTabControl –** набор вкладок**.** Наиболее часто **TTabControl** применяется для динамического создания многооконных интерфейсов совместно с компонентами **TMemo** и **TRichEdit**.
- **TPageControl –** Набор страниц для многостраничного диалогового окна. Весьма распространенный элемент управления в **Windows** приложениях. **TPageControl** позволяет размещать на своих страницах другие элементы управления, обеспечивая быстрый доступ к последним, путем выбора необходимой страницы. Палитра компонентов среды **Delphi** сама представляет собой **PageControl.** Типичными представителями данного элемента являются диалоговые окна выбора параметров в приложениях **Ms Word** и **Ms Excel**, организация построения и доступа к графическим фильтрам и спецэффектам в приложении **Ulead Photo Impact**.
- **TImageList** – компонент содержащий список изображений. Применяется для хранения изображений малых размеров и иконок. Доступ к изображениям осуществляется через их индекс (порядковый номер).

- **TRichEdit** – поле ввода текстовой информации в формате **RTF**. **TRichEdit** подобен компоненту **TMemo,** но обладает значительно большими возможностями, такими как форматирование текста, постраничное его представление и печать. В данном компоненте отсутствует ограничение на объем текстовой информации. Элемент управления **RichEdit,** по своим характеристикам, подобен полю ввода приложения **WordPad OS Windows.**
- **TTrackBar** – бегунок с масштабной линейкой. Используется для управления положением других объектов и интерактивного задания параметров исполняемым процедурам (например: регулировка яркости и контраста образа).
- **TProgressBar** – индикатор. Применяется для отображения процесса. **TUpDown** – управляющий элемент, содержащий кнопки с изображением стрелок.
- **TAnimate** – Анимационное окно, в котором может быть показан немой клип - фильм в формате **AVI** без компрессии. Также компонент **TAnimate** применяется в качестве индикатора процесса.
- **THotKey** – компонент связывающий сочетание клавиш с командой меню.
- **TDataTimePicker** – окно ввода, в котором значения даты и времени могут быть выбраны в открывающемся календаре.
- **TTreeView** – диаграмма древовидной иерархической структуры объектов, является аналогом окна доступа к папкам **Windows** приложения **Проводник**.
- **TListView** – список с колонками для отображения данных. **TListView** представляет собой аналог окна доступа к файлам **Windows** приложения **Проводник**.
- **THeaderControl –** набор заголовков, изменяющих свои размеры.
- **StatusBar** – строка статуса. Представление информации при помощи данного объекта подобно представлению информации в строке статуса **MS Word**.
- **TToolBar** – панель кнопок. **TToolBar** предназначена для создания стандартных панелей инструментов, подобных панели инструментов текстового редактора **WordPad**.
- **TCoolBar** – набор перемещаемых масштабируемых панелей.

2.3 Компоненты страницы System

Компоненты, расположенные на странице System (рис. 4.) предназначены для управления обменом данных между приложениями, управления мультимедийными приложениями, а также для создания аниматоров и симуляторов.

Компоненты, расположенные на странице System

Рис.4

Всего страница содержит 8 компонентов следующего предназначения:

- **Timer –** компонент контроля времени, в котором событие **OnTimer** периодически вызывается через промежуток времени, указанный в свойстве **Interval**. **Timer** применяется при создании аниматоров, симуляторов и программ управления процессами и оборудованием.
- **TPaintBox –** окно для рисования. В обработчиках событий, связанных с управлением мышью передаются относительные координаты положения мыши в **PaintBox**, а не абсолютные координаты формы. Компонент **TPaintBox** является аналогом окна рисования графического редактора **Paint OS Windows.**
- **TMediaPlayer –** служит для управления мультимедийными устройствами (CDROM, MIDI и др.). **MediaPlayer** выполнен в виде панели управления с кнопками **Play**, **Stop**, **Record** и др. Для воспроизведения видео и звука может понадобиться как соответствующее оборудование, так и программное обеспечение. Подключение устройств и установка программ производится в среде **Windows**. Например, для воспроизведения видео, записанного в формате **AVI** сжатым кодеком **MPEG4**, потребуется установить соответствующий драйвер.
- **TOLEContainer** – контейнер, в который могут загружаться или храниться **OLE (***Objects Linked and Embedded***)** объекты. **TOLEContainer** является аналогом **OLE** контейнера **Windows** приложений **Word и Excel.**

- **TDdeClientConv** – служит для установления **DDE (***Dynamic Data Exchange***)** связи с сервером и осуществляет общее управление DDE-связью. Устанавливать связь с DDE-сервером можно как во время дизайна приложения, так и во время его выполнения.
- **TDdeClientItem** – клиент **DDE**, который обеспечивает пересылку данных на сервер и выполнение макросов.
- **TDdeServerConv** – устанавливает связь с клиентом **DDE** и осуществляет управление динамическим обменом данных, также выполняет обработку запросов поступающих от приложений-клиентов на выполнение макроса.
- **TDdeServerItem** – сервер **DDE**, обеспечивающий связь с объектом **DdeServerConv** и определяющий данные, которые передаются по **DDE**, посредством свойств **Text** и **Lines**.

2.4 Компоненты страницы Internet

Страница Internet (рис.5) содержит 14 компонентов предназначенных для создания Internet приложений и WEB браузеров.

Страница Internet

Рис.5

2.5 Компоненты страницы Data Access

Компоненты, расположенные на станице **Data Access** (рис.6) предназначены для доступа к базам данных. Данные компоненты являются невидимыми объектами.

Компоненты доступа к базам данных

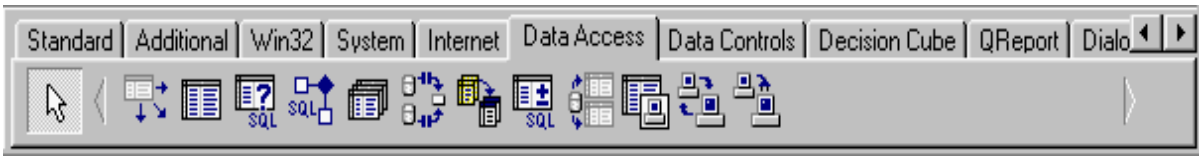

Рис.6

2.6 Компоненты страницы Data Controls

Компоненты, расположенные на странице **Data Controls**, представляют собой элементы управления данными. На странице представлены 15 компонентов. Эти компоненты подобны компонентам, расположенным на страницах **Standard** и **Additional**.

Компоненты управления данными баз данных

Рис.7

2.7 Компоненты страницы Decision Cube

Компоненты, представленные на странице **Decision Cube** предназначены для обеспечения доступа к данным и управления данными.

Компоненты, расположенные на странице Decision Cube

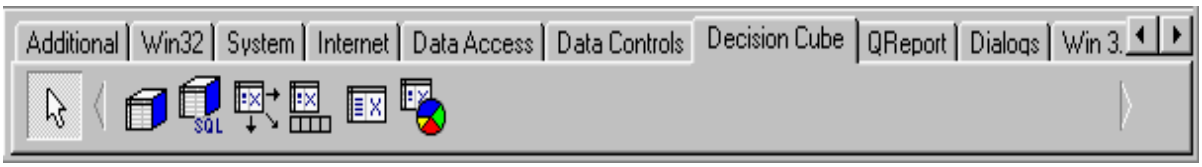

Рис.8

Страница Decision Cube содержит 6 компонентов, три из которых обеспечивают доступ к данным и являются невидимыми объектами, три других представляют собой элементы управления данными. Ниже приводится назначение каждого компонента.

- **TDecisionCube** – многомерное хранилище данных. Куб решений. Невидимый объект.
- **TDecisionQuery** – специализированная форма объекта **TQuery** используемая для определения данных в кубе решений. Невидимый объект.
- **TDecisionSource** – невидимый объект, определяющий текущее положение исходной точки сетки решений или графа решений.

- **TDecisionPivot** – элемент управления, позволяющий изменить размерности или поля куба решений с помощью нажатия кнопок.
- **TDecisionGrid** – компонент представляющий многомерные данные в табличной форме.
- **TDecisionGraph** – компонент, отображающий поля объекта **DecisionCube** в виде динамического графа.

2.8 Компоненты страницы Qreport

На странице **QReport** находится набор из 18 компонентов, предназначенных для быстрой разработки и печати простых отчетов, сложных композитных ленточных отчетов и создания приложений дизайнеров отчетов.

Страница Qreport

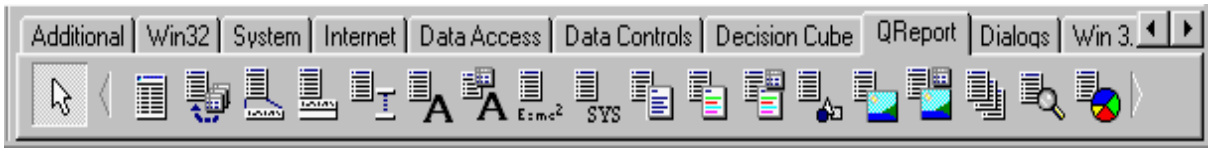

Рис.9

2.9 Компоненты страницы Dialogs

На странице **Dialogs** (рис.10) представлены компоненты вызова и настройки стандартных диалоговых окон операционной системы **Windows**. Внешний вид диалоговых окон зависит от применяемой версии **Windows**.

Компоненты, расположенные на странице Dialogs

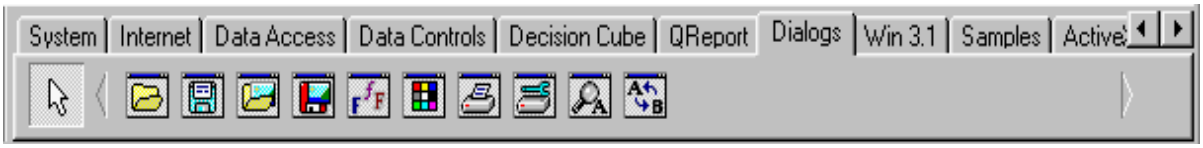

Рис.10

Рассмотрим назначение компонентов в порядке их расположения на странице.

- **TOpenDialog** – вызов окна открытия файла.
- **TSaveDialog** – вызов окна сохранения файла.

- **TOpenPictureDialog** – вызов окна открытия графического файла.
- **TSavePictureDialog** – вызов окна сохранения графического файла.
- **TFontDialog** – вызов окна выбора шрифта.
- **TColorDialog** – вызов окна выбора цвета.
- **TPrintDialog** – вызов окна печати документа.
- **TPrinterSetupDialog** – вызов окна настройки принтера.
- **TFindDialog** – вызов окна поиска текста в строках.
- **TReplaceDialog** – вызов окна поиска текста с заменой текста по образцу.

В компонентах вызова диалоговых окон доступа к файлам, видимость файлов определяется посредством свойства **Filter**. Название диалогового окна можно изменять через свойство **Title**. Выполнение объема диалога и вывод дополнительных сообщений определяются в свойстве **Options**, путем выбора соответствующих пунктов в списке свойства.

Для всех диалоговых компонентов вызов соответствующего диалогового окна осуществляется путем вызова метода (функции) **Execute**. В приведенных ниже примерах показана реализация данного метода.

В примерах 2 и 3 демонстрируется применение компонентов **TOpenPictureDialog** и **TSavePictureDialog** для загрузки графического файла в компонент **TImage** и последующего его сохранения на диске. Имя загружаемого файла определяется значением свойства **FileName**.

Пример 2

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  If OpenPictureDialog.Execute then
  Image.Picture.LoadFromFile(OpenPictureDialog.FileName);
end;
```

Пример 3

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   if SavePictureDialog.Execute then
   Image.Picture.SaveToFile(SavePictureDialog.FileName);
end;
```

2.10 Компоненты страницы Win 3.1

Компоненты, расположенные на странице Win 3.1 (рис.11), соответствуют стандарту и виду окон ранних версий операционной системы **Windows**. В основном они используются для дизайна приложений в стиле **Windows 3.1**. Эти компоненты с успехом могут применяться для создания собственных стилей приложений.

Компоненты, расположенные на странице Win 3.1

Рис.11

Ниже описывается назначение компонентов в соответствии их расположению на странице.

- **TDBLookupList –** Связанный с данными список, содержащий данные полей из другого набора данных.
- **TDBLookupCombo –** Связанное с данными поле со списком, содержащее данные из другого набора данных.
- **TTabSet** – горизонтальные закладки. Обычно используется вместе с **TNoteBook** для создания многостраничных окон. Название страниц можно описывать в свойстве **Tabs**.
- **Outline** – используется для представления иерархических отношений связанных данных. Например – дерево представления папок.
- **TTabbedNotebook** – многостраничный диалог со встроенными закладками.
- **TNotebook** – используется для создания многостраничного диалога, на каждой странице располагается свой набор объектов. Используется совместно с **TTabSet**.
- **THeader** – элемент оформления для создания заголовков с изменяемыми размерами для таблиц.
- **TFileListBox** – специализированный **ListBox**, в котором отображаются файлы из указанной папки (свойство **Directory**). На названия файлов

можно наложить маску, для этого служит свойство **Mask**. Кроме того, в свойстве **FileEdit** можно указать объект **Edit** для редактирования маски.

- **TDirectoryListBox –** специализированный **ListBox**, в котором отображается структура директорий текущего диска. В свойстве **FileList** можно указать **FileListBox**, который будет автоматически отслеживать переход в другую, папку.
- **TDriveComboBox –** специализированный **ComboBox** для выбора текущего диска. Имеет свойство **DirList**, в котором можно указать **DirectoryListBox**, который будет отслеживать переход на другой диск.
- **TFilterComboBox** – специализированный **ComboBox** для выбора маски имени файлов. Список масок определяется в свойстве **Filter**. В свойстве **FileList** указывается **FileListBox**, которому присваивается маска.

С помощью последних четырех компонентов **TFileListBox**, **TDirectoryListBox**, **TdriveComboBox** и **TFilterComboBox** можно построить свой собственный диалог выбора файла, причем для этого не потребуется написать ни одной строчки кода.

2.11 Компоненты страницы Samples

Обычно на странице **Samples** (рис.12) регистрируются свободно распространяемые компоненты, находящиеся в стадии разработки или тестирования. На данной странице представлены 12 образцов компонентов, семь из которых входят в стандартную поставку **Delphi**.

Компоненты, находящиеся на странице Samples

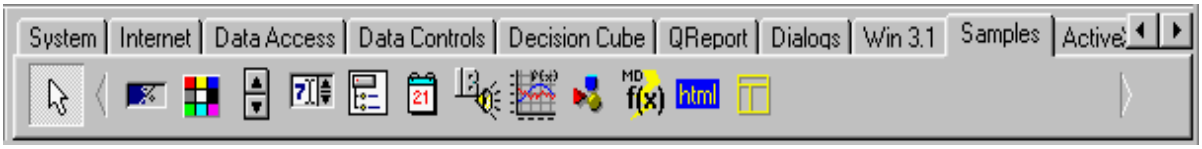

Рис.12

- **TGauge** – индикатор процесса.
- **TColorGrid** – таблица цветов.

- **TSpinButton** – кнопки для дискретного увеличения и уменьшения значений в поле ввода.
- **TSpinEdit** – поле с кнопками пошагового изменения значения ввода.
- **TDirectoryOutline** – структура каталогов текущего диска.
- **TCalendar** – табличный календарь.
- **TIBEventAlerter** – компонент обработки сообщений сервера **InterBase**.

2.11 Объекты страницы ActiveX

Поскольку формат объектов из **MicroSoft Visual Basic** (**VBX**) является своего рода стандартом и существует большое количество библиотек таких объектов, то в среде **Delphi** предусмотрена технология **ActiveX** обеспечивающая совместимость с этим форматом. Объекты данного типа регистрируются на странице **ActiveX** Палитры Компонентов (рис.13).

Объекты, расположенные на странице ActiveX**.**

Рис.13

Зарегистрированные в среде **Windows ActiveX** (**OCX**) объекты можно включить в Палитру Компонентов **Delphi** и использовать их как «родные» компоненты в том числе, выбирать их в качестве *предков* и *наследовать* их свойства и методы.

Ниже приводится перечень объектов, входящих в поставку **Delphi**.

- **Chartfx** – объект, создающий легко модифицированные диаграммы.
- **VSSpell** – визуальный орфографический корректор.
- **F1Book** – полнофункциональная электронная таблица.
- **VtChart** – объект, который позволяет создавать 3D диаграммы.
- **Graph** – конструктор 2D графиков.

ТЕМА 3 ОСНОВНЫЕ ОПЕРАЦИИ С КОМПОНЕНТАМИ СРЕДЫ DELPHI

3.1 Редактирование компонентов в форме

После помещения компонентов в форму пользователь может адаптировать их к требованиям выполняемой разработки. Компоненты можно перемещать, выравнивать, увеличивать или уменьшать их размеры, а также редактировать их свойства.

При установке компонентов в форму рекомендуется заменять их имена более подходящими. Если пользователь присвоит смысловые имена компонентам, то это облегчит читаемость программного кода.

Поскольку компоненты рассматриваются как стандартные элементы управления **OS Windows**, то для их редактирования можно применять все стандартные способы.

Когда пользователь выделяет компонент, то он автоматически становится активным. Заметим, что неотмеченный компонент редактировать нельзя. Чтобы выделить компонент, надо выполнить щелчок мышью прямо на нем в форме или выбрать идентификатор компонента в окне **Object Inspector**. При помощи клавиши [**Tab**] можно поочередно выбирать компоненты в проектировщике форм. Компонент считается выделенным, когда на его сторонах появятся маркеры масштабирования. Чтобы отменить выбор компонента, достаточно выполнить щелчок мышью на пустом месте в проектировщике форм.

3.2 Вырезание, копирование и вставка компонентов

Команды редактирования **Cut**, **Copy** и **Paste** меню **Edit** применяются для компонентов точно так же, как и для текста. Надо помнить следующее: если компонент скопирован из формы в буфер, то его свойства также сохраняются в буфере обмена. Если после этого вставить компонент в форму, используя команду **Paste** меню **Edit** в редакторе кода, в файл модуля будут вставлены свойства этого компонента. Следует учитывать, что программный код обработчиков событий написанный пользователем не вырезается в буфер. Это свойство удобно применять при дизайне, когда требуется перемещение компонента с одного объекта на другой, например, с формы на панель.

3.3 Удаление компонентов

При помощи команды **Delete** меню **Edit** можно удалить отмеченный компонент или отмеченную группу компонентов. Для удаления, также можно использовать клавишу [**Del**]. Команда **Undelete** отменяет ошибочно выполненное действие.

При удалении компонента, с которым пользователь работал, необходимо предварительно удалить все программные коды обработчиков событий, которые ранее были написаны. Если это условие не будет выполнено, то придется вручную удалять во всех секциях модуля объявления и ссылки на удаленный компонент.

ТЕМА 4 УПРАВЛЕНИЕ СВОЙСТВАМИ ВИЗУАЛЬНЫХ КОМПОНЕНТОВ

4.1 Управление свойствами визуальных компонентов в период разработки приложения

Каждый компонент, который пользователь помещает в форму, имеет свое отражение в окне «**Инспектора Объектов**» (**Object Inspector**). Напомним, что «**Инспектор Объектов**» содержит две страницы – **Properties** (Свойства) и **Events** (События). Создание программы в среде **Delphi** сводится к установке компонентов в форму, которая также является компонентом. Форма определяет взаимодействия между установленными в нее объектами. Взаимодействия объектов заключаются в изменениях значений их свойств и адекватных реакций на события этих объектов.

Свойство является важнейшим атрибутом компонента. Для пользователя (программиста) свойство выглядит как простое поле какой-либо структуры, содержащее некоторое значение. Однако, в отличие от простого поля, любые изменения значений свойств компонента приводят к изменению визуального его представления, поскольку свойство *инкапсулирует* в себе *методы* (действия), связанные с чтением и записью в это поле. Свойства служат двум главным целям. Во-первых, они определяют внешний вид формы или компонента, а во-вторых, свойства определяют поведение формы или компонента.

Существует много типов свойств, описывающих объект управления. Эти свойства можно систематизировать по следующим признакам:

- Простые свойства – это те свойства, значения которых являются числами или строками. Например, свойства **Left, Top, Width, Height** принимают целые значения, определяющие положение левого верхнего угла компонента или формы и их ширину и высоту. Свойства **Caption** и **Name** представляют собой строки и определяют заголовок и имя компонента или формы.
- Перечисляемые свойства – это те свойства, которые могут принимать значения из предопределенного набора (списка). Например, свойство типа **Boolean**, может принимать значения **True** или **False**. Другим примером является свойство **Color**, которое может принимать значения **clBlack, clMaroon, clGreen, clNavy** и другие.

Вложенные свойства – это такие свойства, которые поддерживают вложенные значения или объекты. В инспекторе объектов данные свойства представляются кнопкой, обозначенной знаком (+)**,** расположенной слева от названия этих свойств. Имеется два вида таких свойств: множества и комбинированные значения. **Object Inspector** отображает множества в квадратных скобках. Если множество пусто, то оно представляется как: **[ ]**. Установки для вложенных свойств вида ***множество*** обычно имеют значения типа **Boolean**. Наиболее распространенным примером такого свойства является свойство **Style** с вложенным множеством булевых значений. Комбинированные значения отображаются в Инспекторе Объектов как коллекция некоторых величин, каждый со своим типом данных. Некоторые свойства, например, **Font**, для изменения своих значений имеют возможность вызвать диалоговое окно. Для этого достаточно щелкнуть мышью по кнопке, расположенной в правой части строки Инспектора Объектов.

В режиме проектирования манипулирование свойствами осуществляется при помощи «**Дизайнера Форм**» (**Forms Designer**) или, страницы **Properties** «**Инспектора Объектов**». Например, для того чтобы изменить свойства **Height** (высоту) и **Width** (ширину) кнопки, достаточно «зацепить» мышью за любой ее угол и переместить угол в нужную позицию.

4.2 Управление свойствами визуальных компонентов в период выполнения приложения

Среда **Delphi** позволяет легко манипулировать свойствами компонентов как в режиме проектирования (**Design time**), так и в режиме выполнения программы (**Run time**).

В режиме выполнения пользователь имеет возможность не только манипулировать всеми свойствами, отображаемыми в Инспекторе Объектов, но и управлять более обширным их списком.

Все изменения значений свойств компонентов в режиме выполнения должны осуществляться путем прямой записи строк кода. В режиме выполнения невозможно использовать Инспектор объектов. Однако доступ к свойствам компонентов можно довольно легко получить программным путем. Все, что пользователь должен сделать для изменения какого-либо свойства - это написать простую строчку кода аналогичную следующей:

MyComponent.Width := 35;

Такой код устанавливает ширину (**Width**) компонента в значение **35**. Если свойство **Width** компонента не было равно **35** к моменту выполнения данной строки программы, то компонент визуально изменит свою ширину.

В объектно-ориентированном языке **Object Paspal**, лежащим в основе **Delphi**, заложен принцип соответствия визуальных компонентов тем предметам, которые они представляют. Разработчики среды **Delphi** поставили перед собой цель, добиться как можно близкого сходства элемента управления с реальной сущностью. Именно из этого принципа родилось понятие – **Свойства**. Если изменить значения свойств **Width** и **Height** компонента **Button**, то кнопка соответствующим образом изменит свои ширину и высоту, т.е. нет необходимости после изменения свойства **Width** указывать объекту, чтобы он «перерисовал» себя, хотя при обычном программировании именно так и поступают. Свойства – это более чем просто данные, они делают эти данные «живыми», и все это происходит в период дизайна и выполнения приложений. Свойства создают иллюзию, как будто пользователь имеете дело с реальными объектами, а не с их виртуальным (программным) представлением.

Продолжим рассмотрение вопросов связанных с управлением свой-

ствами визуальных компонентов на примере применения графических объектов, Как правило, данные объекты обладают свойствами **Bitmap** и **Canvas**, которое обеспечивают доступ к графическому образу в процессе выполнения программы.

Известно, что в стандартную библиотеку визуальных компонентов среды **Delphi** входит несколько объектов, при помощи которых можно придать своей программе совершенно оригинальный вид. Это компоненты: **TImage**, **TDBImage**, **TShape**, **TBevel**.

Компонент **TImage** позволяет поместить графическое изображение в любое место формы. В него можно загрузить картинку (графический образ) во время дизайна и при выполнении приложения. Картинка должна храниться в файловом формате **BMP** (bitmap), **WMF** (Windows Meta File) или **ICO** (icon). Для работы с базами данных имеется аналог **TImage,** который представлен экземпляром **DBImage,** расположенным на странице **Data Controls «Палитры Компонентов»**. Данный компонент отображает картинку, хранящуюся в поле типа **BLOB** таблицы баз данных.

Известно, что форматов хранения изображений значительно больше тех трех, которые были описаны ранее. Наиболее распространенными являются такие графические форматы как: **PCX, GIF, TIF** и **JPEG**. Для включения в программу изображений в этих форматах нужно либо перевести их в формат **BMP**, либо применять библиотеки третьих фирм, в которых есть аналог компонента **TImage**, понимающий данные форматы.

При проектировании приложений следует помнить, что изображение, помещенное в форму во время дизайна, включается в файл ресурсов и затем компилируется в исполняемый **(EXE)** файл. Вследствие чего такой файл может иметь достаточно большой объем. Как альтернативу можно рассмотреть загрузку картинки во время выполнения приложения. Так как свойство **Picture компонента TImage**, также является объектом, содержащим набор свойств и методов, у свойства **Picture** есть специальный метод **LoadFromFile**, обеспечивающий загрузку изображения. Реализация указанного метода выглядит так:

Image1.Picture.LoadFromFile(FileName);

Важными являются свойства объекта **Center** и **Stretch**. Эти свойства

имеют булев тип. Если свойство **Center** установлено в **True**, то центр изображения будет совмещаться с центром объекта **Image**. Если **Stretch** установлено в **True**, то изображение будет сжиматься или растягиваться таким образом, чтобы заполнить весь объект **Image**.

Другим важным свойством компонента **TImage** является свойство **TBitmap**, позволяющее модифицировать изображение в процессе выполнения приложения. Данное свойство, также представляющее собой объект, применяется в редакторах изображений. При помощи этого свойства можно получить доступ к большому количеству свойств и методов класса **TBitmap**.

Пример 1

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := IntToStr(Image1.Picture.Bitmap.Width);
  Label2.Caption := IntToStr(Image1.Picture.Bitmap.Height);
end;
```

Реализация программного кода при нажатии на кнопку **Button1**

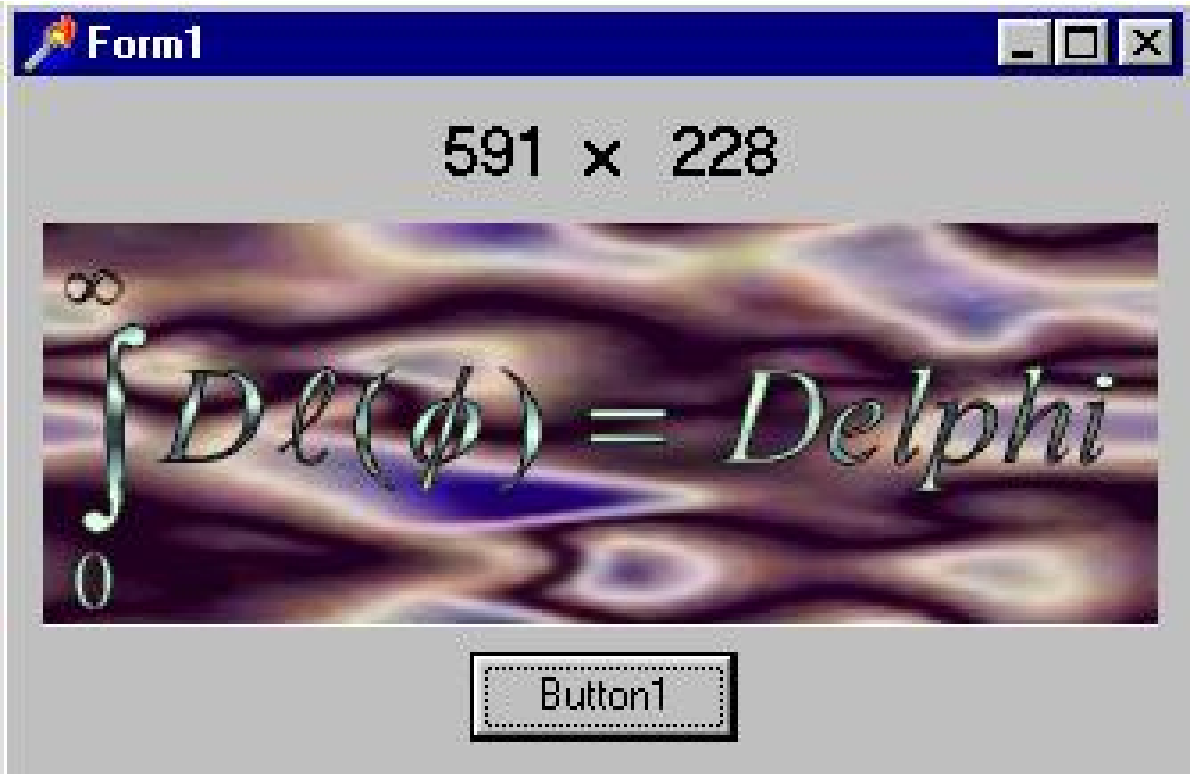

Рис.14

Хотя компонент **TImage** и обладает большим набором свойств и методов, создать на нем мощное приложение достаточно трудно. С другой стороны, на его основе можно создать собственный компонент, наделив его теми свойствами и методами, которые необходимы для профессиональной обработки изображений. В процессе детального рассмотрения этого компонента, после изучения особенностей и методов программно ориентированного программирования и языка **Object Pascal**, прелагается самостоятельно разработать собственный компонент, наделив его дополнительными свойствами и методами.

Компонент **TShape**, позволяющий создавать простые графические образы в форме. Вид необходимого образа выбирается из списка свойства **Shape**. Свойство **Pen** определяет цвет и вид границы образа, а свойство **Brush** задает цвет и вид заполнения образа. Эти свойства можно изменять как во время дизайна, так и во время выполнения приложения.

TBevel – объект для украшения программы, может принимать вид рамки или линии. Он предоставляет меньше возможностей по сравнению с **TPanel**, но не занимает ресурсов. Внешний вид указывается с помощью свойств **Shape** и **Style**.

Известно, что на странице **System** Палитры Компонентов есть объект **PaintBox**, который можно использовать для построения простых приложений вида несложного графического редактора или, например, в качестве среды построения графиков. Никаких ключевых свойств, кроме **Canvas**, объект **PaintBox** не имеет. Этот объект является просто канвой для рисования. Однако у компонента **TPaintBox** есть важная особенность, связанная с тем, что координаты указателя мыши, передаваемые в обработчики соответствующих событий **OnMouseMove** и др., являются относительными. Они передаются как значения координат мыши относительно левого верхнего угла объекта **PaintBox**, а не относительно левого верхнего угла формы.

У ряда других объектов из библиотеки визуальных компонентов также есть свойство **Canvas** (канва), которое предоставляет простой путь для рисования на них. Это объекты такие как: **Bitmap**, **ComboBox**, **DBComboBox**, **DBGrid**, **DBListBox**, **DirectoryListBox**, **DrawGrid**, **FileListBox**, **Form**, **Image**, **ListBox**, **Outline**, **Printer**, **StringGrid**. Свойство **Canvas** является в свою оче-

редь объектом, объединяющим в себе поле для рисования, карандаш **Pen**, кисть **Brush** и шрифт **Font**. Свойство **Canvas** обладает также рядом графических методов: **Draw**, **TextOut**, **Arc**, **Rectangle** и др. Используя свойство **Canvas**, пользователь может воспроизводить в форме любые графические образы такие как: картинки, многоугольники, текст и т.п. без использования компонентов **TImage**, **TShape** и **TLabel** т.е. без использования дополнительных ресурсов. Однако при этом необходимо обрабатывать событие **OnPaint** того объекта, на канве которого выполняются действия. Рассмотрим подробнее свойства и методы объекта **Canvas**.

Основные свойства **Canvas** следующие:

- **Brush** – кисть, является объектом со своим набором свойств.
- **Bitmap** – картинка размером строго 8x8, используется для заполнения (заливки) области на экране.
- **Color** – цвет заливки.
- **Style** - предопределенный стиль заливки.
- **Handle** – данное свойство дает возможность использовать кисть в прямых вызовах процедур **Windows API**.
- **ClipRect** – прямоугольник, на котором происходит графический вывод. Возможно только чтение данного свойства.
- **CopyMode** – свойство определяет, каким образом будет происходить копирование (метод **CopyRect**) на данную канву изображения из другого объекта: один к одному, с инверсией изображения и др.
- **Font** – шрифт, которым выводится текст (метод **TextOut**).
- **Handle** – данное свойство используется для прямых вызовов **Windows API**.
- **Pen** – карандаш, определяет вид линий, как и кисть **Brush** является объектом с набором следующих свойств:
- **Color** – цвет линии.
- **Handle** – для прямых вызовов **Windows API**.
- **Mode** – режим вывода: простая линия, линия с инвертированием, линия с выполнением логической операции «Исключительно или» и др.
- **Style** – стиль вывода: линия, пунктир и др.
- **Width** – ширина линии в точках.

- **PenPos** – текущая позиция карандаша. Карандаш рекомендуется перемещать с помощью метода **MoveTo**, а не прямой установкой данного свойства.
- **Pixels** – двухмерный массив элементов изображения (**pixel**). С его помощью осуществляется доступ к каждой отдельной точке изображения.

4.3 Методы свойства Canvas

Свойство **Canvas** реализует следующие методы рисования простейшей графики: **Arc, Chord, LineTo, Pie, Polygon, PolyLine, Rectangle, RoundRect.** При прорисовке линий используются карандаш **Pen** канвы, а для заполнения внутренних областей – кисть **Brush**.

Методами вывода картинок на канву являются: **Draw** и **StretchDraw,** В качестве параметров указываются прямоугольник и графический объект, на который выводится информация. Метод **StretchDraw** отличается тем, что растягивает или сжимает картинку так, чтобы она заполнила весь указанный прямоугольник.

Методами вывода текста являются: **TextOut** и **TextRect.** При выводе текста используется шрифт **Font** канвы. При использовании **TextRect** текст выводится только внутри указанного прямоугольника. Длину и высоту текста можно узнать с помощью функций **TextWidth** и **TextHeight.**

ТЕМА 5 СОБЫТИЯ И ОБРАБОТЧИКИ СОБЫТИЙ. НАПИСАНИЕ ПРОГРАММНОГО КОДА.

5.1 Виды событий

Все события в среде **Delphi** можно разделить на две категории, а именно: события, обусловленные действиями пользователя (пользовательские события), и программно-управляемые события (обычные события). Процедуры обработки пользовательских событий составляют главную часть написанного программистом текста программы. Они обеспечивают интерактивное взаимодействие приложения и пользователя. В среде **Delphi** для этой цели применяются предварительно определенные обработчики событий.

К программно-управляемым событиям относятся события: активизации и завершения работы приложения, изменения состояния компонентов и другие. Они являются косвенным результатом действия пользователя или действиями самой программы.

5.2 Виды событий, обусловленные действиями пользователя

Пользовательские события (**user events**) возникают в результате выполненных действий, например, выбора команды меню. Задача разработчика приложения состоит в создании программы обработки данного события. Помимо известных стандартных и дополнительных обработчиков событий компонентов другими важными являются события, такие как события мыши, операции **Drag & Drop** (перенести и оставить) и события клавиатуры. Рассмотрим основные из них.

5.3 Стандартные обработчики событий

Для многих компонентов в среде **Delphi** уже определен стандартный обработчик событий. Чаще всего имена стандартных обработчиков событий располагаются в самой верхней строке страницы **Events** окна **Object Inspector**. Когда пользователь открывает страницу **Events**, стандартный обработчик события становится активным.

Можно создать процедуру обработки стандартного события, выполнив двойной щелчок левой клавишей мыши на компоненте в форме. При этом активизируется редактор кода.

5.4 Нестандартные обработчики событий

Для остальных обработчиков событий следует использовать страницу **Events** инспектора объектов, чтобы установить, какие обработчики событий могут быть использованы для данного компонента. Если выполнить двойной щелчок левой клавишей мыши на имени обработчика события или нажать клавиши [**Ctrl + Enter**] в окне инспектора объектов, среда **Delphi** создаст пустую процедуру обработки события, после чего редактор кода получит фокус и курсор будет помещен в начало процедуры обработки события.

5.5 Связанные процедуры с обработчиком события

Процедура обработки события, может быть связана с обработчиком события. Если в модуле формы уже определены процедуры обработки событий, то на странице **Events** для обработчиков событий, имеющих тот же тип, можно открыть список со всеми процедурами, которые могут быть связаны с этим обработчиком события. При выборе процедуры обработки события в поле со списком программный код не дублируется. Таким образом, для одних и тех же обработчиков событий различных компонентов можно использовать один и тот же программный код.

5.6 Написание программного кода

Среда **Delphi** автоматически объявляет каждый компонент, включаемый в модуль формы. Объявление, например, трех компонентов проекта в модуле выглядит следующим образом:

```
Type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    CheckBox1: TCheckBox;
  Private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Компоненты **Label1**, **Edit1** и **CheckBox1** объявлены в **published** секции класса **TForm1**. Компоненты всегда объявляются, как только они помещаются в форму. Форма является ***владельцем*** компонентов.

5.7 Создание процедуры обработки событий

Допустим, в форме **Form1** находится компонент **Label1**, задачей которого является вывод сообщения о том, что в компонент **Edit1** должно быть введено число, при условии выбранной опции **CheckBox1**. Тогда программу следует разрабатывать следующим образом. Когда выполняется

щелчок мыши на объекте **CheckBox1**, надпись в **Label1** изменяется на "**Введите число**". Щелчок на **CheckBox1** выполненный пользователем, является для приложения ***событием***. Это событие вызывает стандартный обработчик **OnClick** компонента **CheckBox1** находящийся в первой строке страницы **Events** инспектора объектов. Чтобы создать процедуру обработки события **OnClick**, достаточно выполнить двойной щелчок на поле рядом с именем события. В ответ на это активизируется редактор кода, а курсор помещается в начало строки, где следует вставить необходимый программный код. Такой подход иллюстрируется в примере 1.

Пример 1

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  if CheckBox1.Checked = true then
    Label1.Caption := 'Введите число';
  else
    Label1.Caption := ' ';
end;
```

Если во время выполнения программы будет выполнен щелчок мышью по опции **CheckBox1,** то выводится сообщение: «Введите число». Приложение выполнит процедуру: **Form1.CheckBox1Click(Sender: TObject**).

При создании процедур обработки событий среда **Delphi** самостоятельно создает идентификаторы данных методов.

5.8 Совместно используемые процедуры обработки событий

Для каждого события, на которое должно реагировать приложение, можно написать свою процедуру. Но можно написать и одну процедуру, затем связать ее с различными обработчиками событий и таким образом позволить нескольким обработчикам событий использовать эту процедуру совместно.

Это имеет смысл в ситуациях, когда различные действия пользователя должны вызывать одну и ту же реакцию.

5.9 События мыши

Как известно, без устройства типа мыши трудно представить себе работу приложений в среде **Windows**. В настоящее время мыши имеют, как правило, две кнопки и могут содержать колесо прокрутки. Такую конструкцию можно считать стандартом для мыши. Таким образом, при разработке приложения можно исходить из того, что пользователь в любой ситуации может применить как левую, так и правую кнопки мыши. Левая кнопка мыши используется как основная, а правая как дополнительная, поэтому в дальнейшем под нажатием на кнопку мыши будем понимать нажатие на левую кнопку.

Когда пользователь перемещает мышь по коврику, то на экране перемещается изображение указателя положения мыши, которое называется **Mouse Cursor** (указатель мыши). Указатель мыши может выглядеть поразному. Разработчик выбирает, какой вид указателя и в какое время он не изменяется при выполнении того или иного действия. Это выполняется с помощью свойства **Cursor**. Изображение указателя мыши зависит от его положения относительно элемента управления, которому присвоено соответствующее значение свойства **Cursor**.

Существуют пять видов действий с устройством типа мышь, на которые реагирует приложение:

- Нажатие кнопки мыши (**MouseDown**).
- Отпускание кнопки мыши (**MouseUp**).
- Перемещение мыши (**MouseMove**).
- Щелчок (**Click**).
- Двойной щелчок (**DblClick**).

Первые три действия выполняются исключительно с устройством типа мышь, четвертое действие может быть выполнено не только с помощью мыши, но и путем нажатия клавиши на клавиатуре, например **Enter**.

Обработчики событий **OnMouseDown** и **OnMouseUp** имеют тип **TMouseEvent**. Обработчик события **TMouseMove** имеет тип **TMouseMoveEvent**. Когда **Delphi** приложение получает сообщение о том, что пользователь произвел какие-либо действия с мышью, то вызывается соответствующая процедура обработки события. В эту процедуру переда-

ются параметры, которые содержат дополнительную информацию о возникшем событии. Например, параметр **Button**, передаваемый в процедуру **TMouseEvent**, содержит информацию о том, какая кнопка мыши была нажата. Параметры, используемые в процедурах типа **TMouseEvent** и **TMouseMoveEvent**, имеют следующее назначение:

- **Sender** – объект на который воздействует пользователь с помощью мыши.
- **Button** – кнопка мыши которая была нажата: **mbLeft**, **mbMidle** или **mbRight**.
- **Shift** – состояние клавиш [**Ctrl**], [**Alt**] и [**Shift**] в момент действия мыши.
- **X**, **Y** – экранные координаты точки, где произошло событие.

Обработчик события **OnMouseDown**, вызывается, когда пользователь нажимает кнопку мыши, при условии, что ее указатель находится на элементе управления. Объект, на котором находится указатель, получает сообщение об этом событии, и выполняется процедура обработки события для этого объекта, если такая процедура была определена. Обработчик события **OnMouseDown** формы**,** приведенный в примере 2. демонстрирует вывод круга диаметром 10 пикселей и текста, содержащего значения координат точки, в том месте, где произошло событие. Событием является нажатие кнопки мыши в пустой форме. Выполнение примера показано на рисунке 15.

Пример 2

```
procedure TForm1.FormMouseDown(Sender: TObject;
            Button: TMouseButton; Shift: TshiftState; X, Y: Integer);
begin   Canvas.Ellipse(X,Y,X-10,Y-10);
   Canvas.TextOut(X, Y, 'X='+IntToStr(X)+' Y='+IntToStr(Y));
end;
```

Выполнение процедуры обработки события «нажатие кнопки мыши»

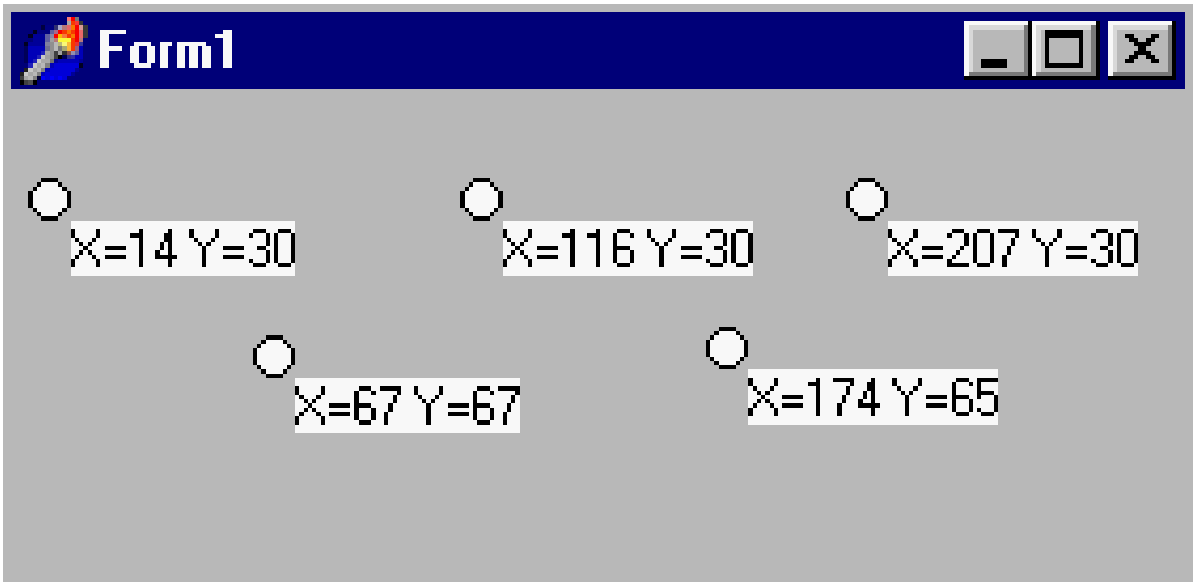

Рис 15

Обработчик события **OnMouseUp** вызывается, когда пользователь отпускает нажатую кнопку мыши. Обычно сообщение об этом событии передается объекту, на котором находится указатель мыши в момент нажатия кнопки. Приведенные ниже процедуры (примеры 3, 4) демонстрируют результат действия пользователя при нажатии и отпускании кнопки мыши в разных местах формы. Реализация указанных процедур, в период выполнения приложения, приводится на рисунке 16.

Пример 3

```
procedure TForm1.FormMouseDown(Sender: TObject;
            Button: TMouseButton;  Shift: TshiftState; X, Y: Integer);
begin
   Canvas.Brush.Color := clLime;   Canvas.Ellipse(X,Y,X-10,Y-10);
   Canvas.Brush.Color := clWhite;
   Canvas.TextOut(X, Y, 'X=' + IntToStr(X) + ' Y=' + IntToStr(Y) +
   ' – Координады нажатой кнопки');
end;
```

Пример 4

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouse-
Button;
                Shift: TshiftState; X, Y: Integer);
begin
  Canvas.Brush.Color := clRed;  Canvas.Ellipse(X, Y, X-10,Y-10);
  Canvas.Brush.Color := clWhite;
  Canvas.TextOut(X, Y, 'X=' + IntToStr(X) + ' Y=' + IntToStr(Y) +
 ' – Координады отпущеной кнопки');
end;
```

Реализация процедур, приведенных в примерах 3, 4

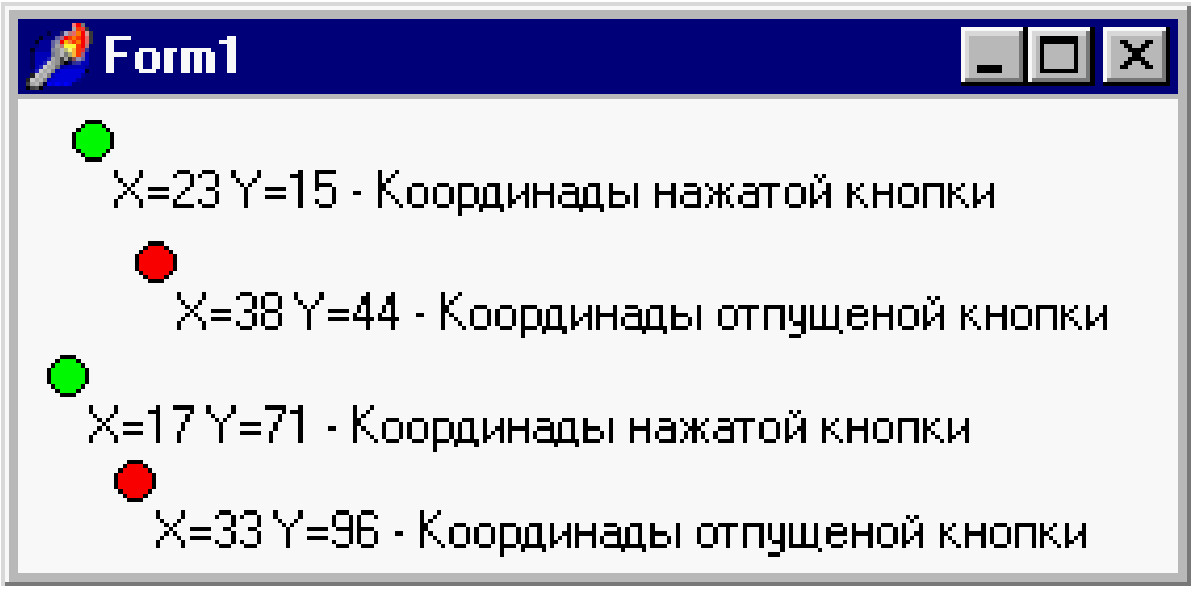

Рис.16

Обработчик события **OnMouseMove** периодически вызывается при перемещении пользователем указателя мыши над элементом управления. В процедуре обработки события **OnMouseMove** значение параметра **Shift** зависит от того, какая клавиша клавиатуры [**Alt**], [**Shift**] или [**Ctrl**] была нажата, когда пользователь перемещал мышь. Параметры **X** и **Y** содержат координаты текущего положения указателя мыши. Обращаем внимание на то, что обработчик события вызывается без нажатой кнопки мыши. Если надо реагировать на перемещение мыши при нажатой кнопке, то следует ввести в модуль дополнительную программную строку для определения

состояния кнопки мыши. Для этого следует включить в описание класса формы поле типа **Boolean**. Пример 5 реализует объявление такого поля.

Пример 5

```
type
  TForm1 = class(TForm)
    Procedure FormMouseDown(Sender: TObject; Button: TmouseButton;
                  Shift: TshiftState; X, Y: Integer);
    Procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
                  Shift: TshiftState; X, Y: Integer);
    Procedure FormMouseMove(Sender: TObject;
                  Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }
  public
  MouseBtn : boolean;
    { Public declarations }
  end;
```

Наряду с объявлением этого логического поля, основное значение имеет место в программе, где поле **MouseBtn** будет получать значение **True**, если кнопка мыши нажата и значение **False** – если она отпущена. Для этого следует использовать обработчики события **OnMouseDown** и **OnMouseUp**. В примерах 6 и 7 приводятся программные строки обработчиков событий **OnMouseDown** и **OnMouseUp**, в которых устанавливается значение поля **MouseBtn**. Также в этих процедурах изменяются свойства канвы для рисования в форме разными цветами при нажатии и отпускании клавиши мыши.

Пример 6

```
procedure TForm1.FormMouseDown(Sender: Tobject; Button:
                TMouseButton;   Shift: TshiftState; X, Y: Integer);
begin
   MouseBtn := true;  Canvas.Pen.Width := 3;
```

```
    Canvas.Pen.Color := clBlue;    Canvas.Brush.Color := clLime;
    Canvas.Ellipse(X,Y,X-10,Y-10);      Canvas.Brush.Color := clWhite;
    Canvas.TextOut(X, Y, 'X=' + IntToStr(X) + ' Y=' + IntToStr(Y) +
    ' – Координады нажатой кнопки');
    Canvas.MoveTo(X,Y);
end;
```

Пример 7

```
procedure TForm1.FormMouseUp(Sender: Tobject; Button: TmouseButton;
                Shift: TshiftState; X, Y: Integer);
begin
  MouseBtn := false;  Canvas.Brush.Color := clRed;
  Canvas.Ellipse(X,Y,X-10,Y-10);  Canvas.Brush.Color := clWhite;
  Canvas.TextOut(X, Y, 'X=' + IntToStr(X) + ' Y=' + IntToStr(Y) +
  ' - Координады отпущеной кнопки');
end;
```

Программные строки обработчика события **OnMouseMove** реализуют процесс рисования в форме при перемещении мыши, если нажата клавиша мыши (пример 8).

Пример 8

```
procedure TForm1.FormMouseMove(Sender: Tobject; Shift: TshiftState; X,
    Y: Integer);
Begin
   if MouseBtn = true then Canvas.LineTo(X,Y);
end;
```

На рисунке 3 показано выполнение программы нажатия, перемещения и отпускания клавиши мыши в форме.

Выполнение программы

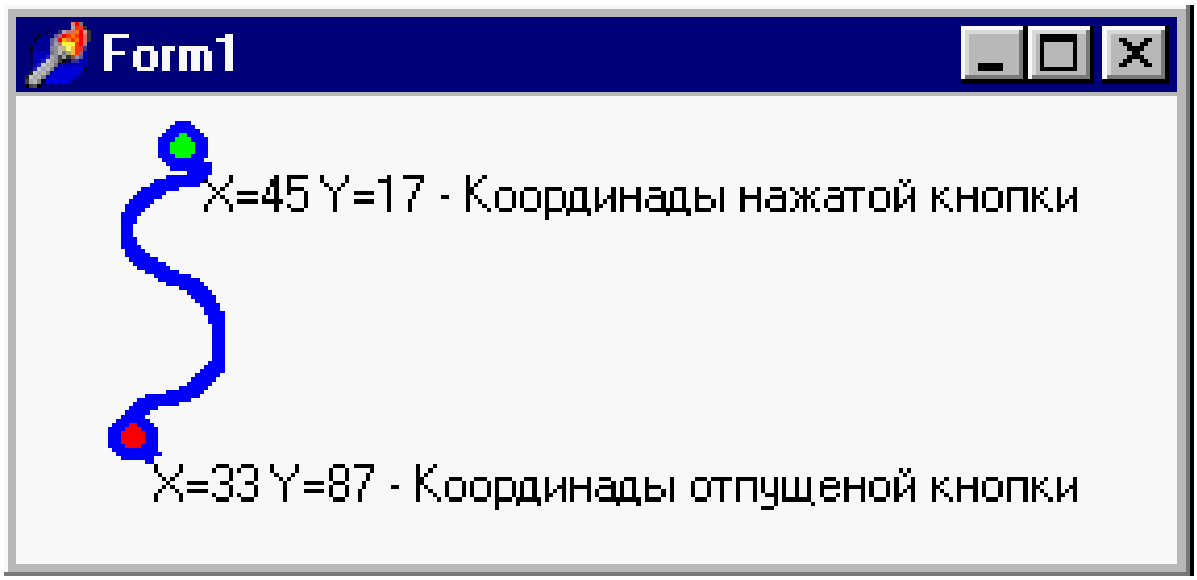

Рис.17

События, которые представляют щелчок мыши на компоненте (рис.17), называются событиями щелчка (**click events**). Для трех из них имеются предварительно определенные обработчики событий: **OnClick**, **OnClickCheck** и **OnDblClick**. Событие **Click** уже неоднократно рассматривалось в ранее приведенных примерах и индивидуальных заданиях, а также самостоятельно разрабатывался программный код для обработки события **OnClick**. Теперь более подробно рассмотрим данное и другие события щелчка.

Все три вышеуказанные события щелчка имеют тип **TNotifyEvent**.

TnotifyEvent = **procedure** (Sender: TObject) of **object**;

Обработчик события **TNotifyEvent** относится к типу обработчиков событий, не имеющих других параметров, кроме параметра **Sender**. Обычно эти события сообщают компонентам лишь то, что определенное событие произошло.

Не все компоненты, обладающие событиями **OnClick**, имеют обработчик события **OnDblClick**. Двойной щелчок на компоненте, для которого определена процедура обработки события **OnClick** и который не имеет обработчика события **OnDblClick**, приводит к вызову **OnClick**. Это означает, что в данном случае двойной щелчек дает тот же эффект, что и одиночный. Обработчик события *OnClick* вызывается, если пользователь вы-

полняет следующие действия:

- Нажимает и отпускает кнопку мыши в тот момент, когда указатель мыши находится на компоненте.
- Отмечает элемент таблицы, списка или поля со списком с помощью кнопок, обозначенных стрелками.
- Нажимает клавишу [**Enter**], когда активная форма содержит предварительно выбранную кнопку по умолчанию.
- Нажимает клавишу [**Space**], когда активна какая-либо кнопка или опция.
- Нажимает клавишу [**Esc**], когда активная форма содержит кнопку **Cancel**.
- Выполняет щелчок на свободном месте формы или на неактивном компоненте.

Кроме того, обработчик события **OnClick** вызывается в следующих случаях, когда значение свойства **Checked** опции было установлено и при вызове метода **Click** кнопки или меню.

5.10 Операции Drag & Drop

При помощи операций **Drag & Drop** пользователи могут редактировать компоненты в форме. При выполнении этих операций имеют значения три основных аспекта, связанные с определением возможности перемещения элементов:

- Начало перемещения.
- «Оставление» элементов.
- Конец перемещения.

Процесс перемещения элементов управления, зависит от ряда условий, которые определяются в программе. Каждый элемент управления имеет свойство **DragMode**, при помощи которого можно управлять началом перемещения элемента. Это свойство может принимать значения **dmMamual** и **dmAutomatic**. Если свойство **DragMode** имеет значение **dmAutomatic**, то элемент управления можно перемещать, когда пользователь нажимает кнопку мыши. Если это свойство имеет значение **dmMamual**, то элемент управления можно перемещать лишь после того,

как будет вызван метод **BeginDrag**. Данный метод использует параметр **Immediate**. Если **Immediate** присвоено значение **False**, то процесс перемещения начинается лишь после того, когда пользователь немного передвинет мышь и ее указатель приобретет вид, определенный значением свойства **DragCursor**. Если параметру **Immediate** присвоено значение **True**, то перемещение начинается сразу. Вызов **BeginDrag(False)** позволяет элементу управления обрабатывать события щелчка, не начиная процесс перемещения. Например, прежде чем вызвать метод **BeginDrag**, можно сначала проверить, какая кнопка мыши была нажата пользователем.

Когда пользователь перемещает какой-либо элемент через определенный компонент, то этот компонент вызывает обработчик события **OnDragOver**. Если данный элемент управления может принять переносимый элемент, форма указателя мыши изменяется, так как изменяется значение свойства **DragCursor**. Изменяя вид указателя мыши, элемент управления показывает, может ли он принять переносимый элемент. Наряду с этим, следует также определить, что произойдет с элементом после того, когда пользователь отпустит кнопку мыши. Для этого необходимо определить процедуру обработки события **OnDragDrop** элемента управления, который принимает переносимый элемент. Обработчик события **OnDragOver** содержит следующие параметры:

- **Source** - объект, с которого начинается операция перемещение. Этот объект не может являться перемещаемым объектом.
- **Sender** - объект, через который производится перемещение.
- **X, Y** – экранные координаты в пикселях.
- **State** – состояние перемещаемого объекта по отношению к объекту, через который он перемещается.
- **Accept** – этот параметр определяет, может ли данный компонент принять переносимый объект.

Процедура обработки события **OnEndDrag** определяет, что должно произойти, когда процесс перемещения завершается. Важнейшим параметром этой процедуры является параметр **Target**, который указывает, какой элемент управления принял объект. Параметры **X** и **Y** – это координаты указателя мыши относительно границ принимающего элемента управления.

5.11 События клавиатуры

Рассматривая события клавиатуры, исходят из того, что не играет роли, появляется или нет видимый символ на экране в результате нажатия клавиши. Событие клавиатуры генерируется, как только клавиша была нажата или отпущена. В обоих случаях приложение получает от среды **Windows** сообщение о нажатии клавиши. Но при детальном рассмотрении данного вопроса, конечно, имеет значение, была ли нажата клавиша с управляющим символом или клавиша с читаемым символом. Следовательно, операционная система **Windows** всегда посылает в приложение сообщение, состоящее из двух частей: сообщение о нажатии клавиши (**Keystroke Message**) и сообщение о символе (**Character Message**). Это отражено в наличии двух обработчиков событий, которые **Delphi** использует для определения реакции на нажатие клавиши. В **Delphi** имеются обработчики событий **OnKeyDown** и **OnKeyUp**, которые вызываются при нажатии и отпускании клавиши клавиатуры, а также обработчик события **OnKeyPress**, который связан с читаемым символом. Естественно нет необходимости реагировать на каждое нажатие клавиши, так как среда **Windows** обрабатывает большую часть нажатий клавиш, например таких, которые определены для команд меню в сочетании с клавишей [**Alt**]. Для различных системных функций, которые могут быть вызваны при помощи клавиатуры, **Windows** располагает стандартными алгоритмами обработки данных событий. Среда **Delphi** также обрабатывает нажатия клавиш, например тех, которыми пользователь может редактировать текст в поле ввода или в другихэлементах управления. Разработчик должен самостоятельно предусматривать перехват нажатий тех клавиш, для которых он установил в приложении определенные функции. События клавиатуры для этих клавиш должны быть обработаны в программном коде приложения.

В среде **Delphi** используется основной принцип, заключающийся в том, что сообщение о событии получает элемент управления, который в данный момент активен для ввода. В примере 9 приводится программные сроки обработчика события нажатия клавиши **F6**, выполнение которых приводит к изменению масштаба картинки, загруженной в **Image1**. Для того, чтобы приведенный метод исполнялся необходимо свойству

KeyPreview формы присвоить значение **True**.

Пример 9

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TshiftState);
begin
   if Key = VK_F6 then Image1.Align := alClient;
end;
```

5.12 Перехват событий клавиатуры

Для форм в среде **Delphi** определено свойство **KeyPreview**, которое может быть установлено таким образом, что события клавиатуры будут сначала проверяться, и обрабатываться самой формой, после чего они могут быть обработаны активным элементом управления. Если данное свойство имеет значение **False**, события клавиатуры поступают непосредственно к тому элементу управления, который активен для ввода. Если же этому свойству присвоено значение **True**, то события клавиатуры сначала будет получать форма.

Наряду с приведенным выше механизмом перехвата событий клавиатуры, существуют еще три других уровня, на которых могут быть приняты и обработаны события клавиатуры. К ним относятся: *Уровень приложения*, *Уровень сочетания клавиш и Уровень компонента*. Уровень приложения использует обработчик события **Application.OnMessage**. При использовании этого события сообщения **Windows** можно перехватить раньше, чем его получат окна приложения. Если в программе определено сочетание клавиш, например, для команды меню, то событие, вызванное нажатием этих клавиш, будет получено и обработано данным компонентом, а не активным элементом управления окна. Если определены процедуры обработки событий клавиатуры, для какого либо компонента, то компонент, активный для ввода, будет перехватывать сообщения о событиях нажатия клавиш.

5.13 Обработчики событий клавиатуры

Обработчики событий **OnKeyDown** и **OnKeyUp** применяются, когда требуется перехватить сочетания управляющих и символьных клавиш, таких как [**Shift + клавиша**], [**Ctrl + клавиша**] и [**Alt + клавиша**]. Также данные обработчики событий используются для функциональных клавиш, которые не содержат **ASCII** символы. При выборе между этими двумя обработчиками событий действует следующие правила:

- Обработчик события **OnKeyDown** рекомендуется использовать для клавиш управления курсором, функциональных клавиш и специальных клавиш, таких как [**Ins**] и [**Del**].
- Обработчик события **OnKeyDown** и **OnKeyUp** целесообразно использовать, когда в промежутке между событиями требуется запустить фоновый процесс.

Обработчики событий имеют тип **TKeyEvent**:

TkeyEvent = **procedure** (Sender: TObject; var Key: Word;
Shift: TShiftState);

Параметр **Key** – это код клавиши. Параметр **Shift** описывает состояние управляющих клавиш:

- **ssShift** - нажата клавиша [**Shift**].
- **ssAlt** – нажата клавиша [**Alt**].
- **ssCtrl** - нажата клавиша [**Ctrl**].
- **ssLeft** – нажата левая кнопка мыши.
- **ssRight** – нажата правая кнопка мыши.
- **ssMiddle** – нажата средняя кнопка мыши.
- **ssDouble** – выполнен двойной щелчок.

Процедуры, приведенные в примере 10, демонстрируют изменение заголовка формы, при условии нажатой и отпущенной клавиши **Z**.

Пример 10

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TshiftState);
begin
   if chr(Key) = 'Z' then  Form1.Caption := 'Нажата клавиша Z';
end;
```

```
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
                                                Shift: TshiftState);
begin
  if chr(Key) = 'Z' then  Form1.Caption := 'Отпущена клавиша Z';
end;
```

Программная строка обработчика собыбытия **OnKeyPress** (пример 11) возвращает отдельный символ из набора **ASCII**, введенный пользователем:

Пример 11

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  Form1.Caption := Key;
end;
```

5.14 Программно-управляемые события

Прямое взаимодействие между пользователем и программой происходит при помощи описанных ранее событий. Кроме этого, существуют и другие обработчики событий. Конкретные типы таких обработчиков событий присущи каждому отдельному компоненту и их количество очень велико. Рассмотрим наиболее часто используемые обработчики событий.

Обработчик события **OnChange**, вызывается, когда компонент или объект изменяется. Пример 11 демонстрирует добавление в список изменений в тексте, содержащемся в поле ввода.

Пример 11

```
Procedure TForm1.Edit1Change(Sender: TObject);
begin
  ListBox1.Items.Add(Label1.Caption + ' - '+ Edit1.Text);
end;
```

Обработчик события **OnEnter**, вызывается, когда оконный элемент управления становится активным для ввода. В примере 12 поле ввода **Edit1** окрашивается желтым цветом, когда становится активным.

Пример 12

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
   Edit1.Color := clYellow;
end;
```

Обработчик события **OnActive** для приложения **Application** вызывается, если приложение становится активным.

- Обработчик события **OnDeactive** для приложения **Application** вызывается, когда пользователь переключается с данного приложения на другое.
- Обработчик события **OnException** вызывается, когда в приложении возникает исключительная ситуация.
- Обработчик события **OnHelp** вызывается, когда приложение получает запрос на вызов справки.
- Обработчик события **OnMinimize** вызывается, когда приложение сворачивается в значок.
- Обработчик события **OnRestore** вызывается, если ранее свернутое в значок приложение восстанавливает первоначальные размеры своего окна.
- Обработчик события **OnIde** периодически вызывается, когда приложение находится в режиме ожидания.
- Обработчик события **OnHint** вызывается, когда пользователь помещает указатель мыши на компонент, при условии, что свойство компонента **ShowHint** установлено в состояние **True** и свойство **Hint** содержит текст подсказки.

Обработчик события **OnMessage** вызывается, когда приложение получает сообщение операционной системы **Windows**. Включив в программу процедуру обработки события **OnMessage**, можно вызывать другие процедуры, которые будут отвечать на поступившие сообщения. Если приложение не располагает специальными процедурами обработки поступившего сообщения, это сообщение обрабатывается средой **Windows**.

Обработчик события **OnActiveFormChange** вызывается, когда активизируется новая форма. Значение свойства **ActiveForm**, компонента **TScreen** указывает, какая форма стала активной.

Література

1. Р. Боас, М. Фервай, Х. Гюнтер, Delphi 4 Полное Руководство, – Киев.: BHV, 1998. – 448с.
2. Developer’s Guide for Delphi 3, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
3. Developer’s Guide for Delphi 5, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
4. Object Pascal Language Guide, Borland Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249
5. Анталогия Delphi, http://www.Torry.ru

ЗМІСТ

Навчальне видання

Швачич Геннадій Григорович
Овсянніков Олександр Васильович
Кузьменко Вячеслав Віталійович
Нєчаєва Наталія Іванівна

Прикладне програмне забезпечення

Конспект лекцій

Тем. план 2007, поз. 153

Підписано до друку 06.02.07. Формат 60х84 1/16. Папір друк. Друк плоский.
Облік.-вид. арк. 2,88. Умов. друк. арк. 2,84 Тираж 100 пр. Замовлення №11

Національна металургійна академія України
49600, Дніпропетровськ – 5, пр. Гагаріна, 4

Редакційно – видавничий відділ НМетАУ