

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ**

**Г. Г. Швачич, О. В. Овсянніков, Л.М.Петречук**

**Методичні вказівки до виконання лабораторних робіт з дисципліни**

**Прикладне програмне забезпечення**  
**для студентів спеціальностей 6.020105 «Документознавство та інформаційна**  
**діяльність»**

**Дніпропетровськ НМетАУ 2012**

## Лабораторная работа №1

Тема: Интегрированная среда разработки Delphi

Цель работы: изучение основных составляющих (окон) среды Delphi

Интегрированная среда разработки Delphi (Delphi IDE) является многооконной системой. Она включает в себя все необходимое для быстрой разработки Windows-приложений, и может гибко настраиваться.

Delphi имеет некоторый стандартный, предусмотренный разработчиками вид, в котором она предстает вам при первом запуске. В таком "стандартном" варианте среда Delphi имеет следующие окна (рис.1):

- 0 - окно связи с Inprise (Borland)
- 1 - главное окно (Delphi 7 - Project1),
- 2 - окно дерева объектов (Object TreeView),
- 3 - совмещенное окно редактора кода и проводника кода (на заднем плане, под формой Form1).
- 4 - окно конструктора форм (Form1),
- 5 - окно инспектора объектов (Object Inspector).

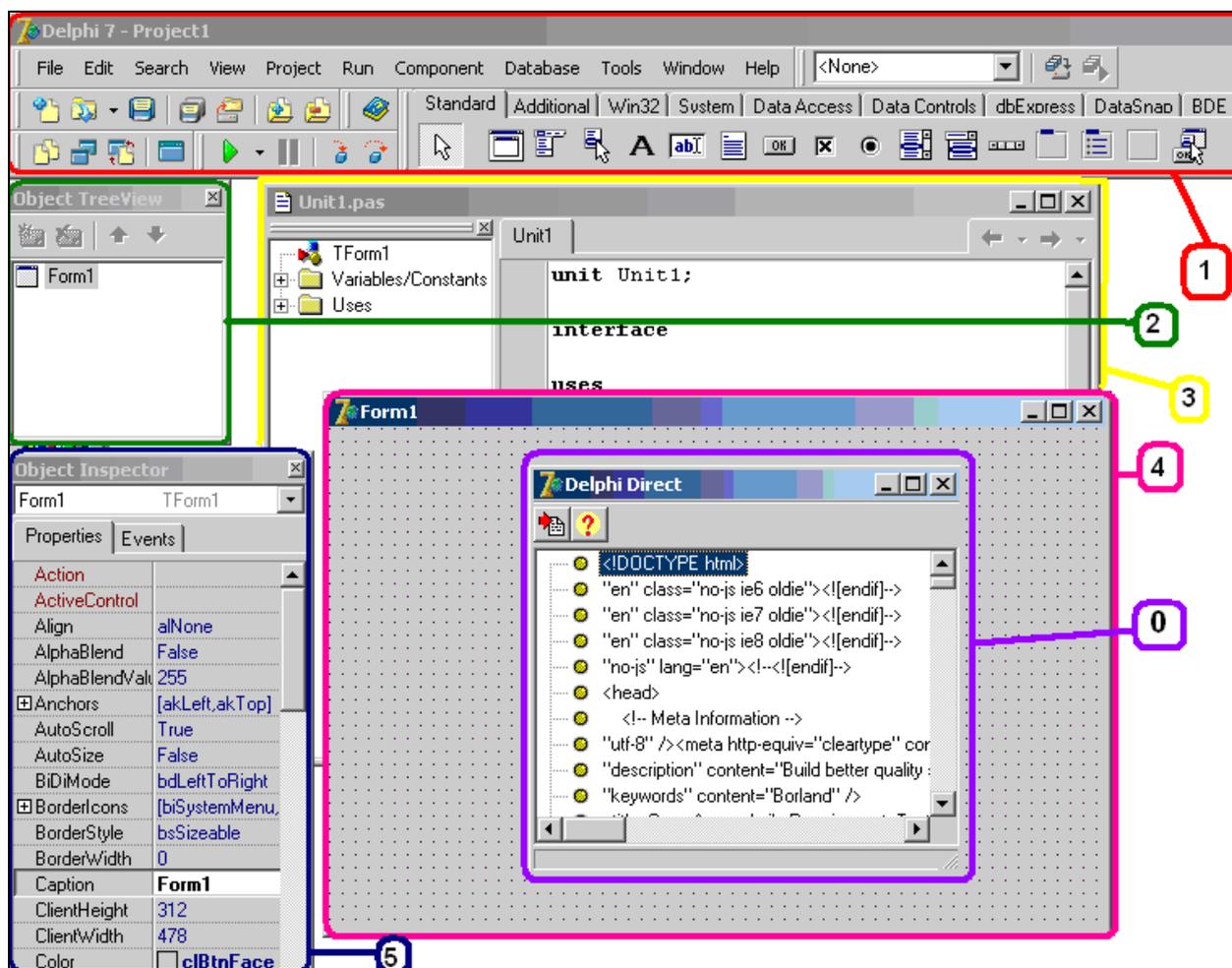


Рис.1. Вид Delphi 7 IDE по умолчанию

**ОКНО СВЯЗИ с Inprise (Borland) (0).** При первом запуске Delphi поверх всех окон появится окно связи с Inprise -0. С помощью этого окна вы сможете получить доступ к Web-страницам корпорации Inprise для просмотра самой свежей информации о корпорации и ее программных продуктах, копирования дополнительных файлов, чтения ответов на наиболее

часто задаваемые вопросы и т. д. При повторных запусках Delphi это окно появляется автоматически с некоторой периодичностью, определяемой настройками на странице окна *Tolls* → *Environment Options*, связанной с закладкой *Delphi Direct*. Вы также сможете его вызвать в любой момент с помощью опции *Help* → *Delphi Direct* главного меню.

**ГЛАВНОЕ ОКНО (1)** осуществляет основные функции управления проектом создаваемой программы. Это окно всегда присутствует на экране и занимает его самую верхнюю часть. Оно несет в себе элементы, которые всегда должны быть под рукой у программиста.

В главном окне располагаются: *главное меню Delphi*, *панели инструментов* и *палитра компонентов*.

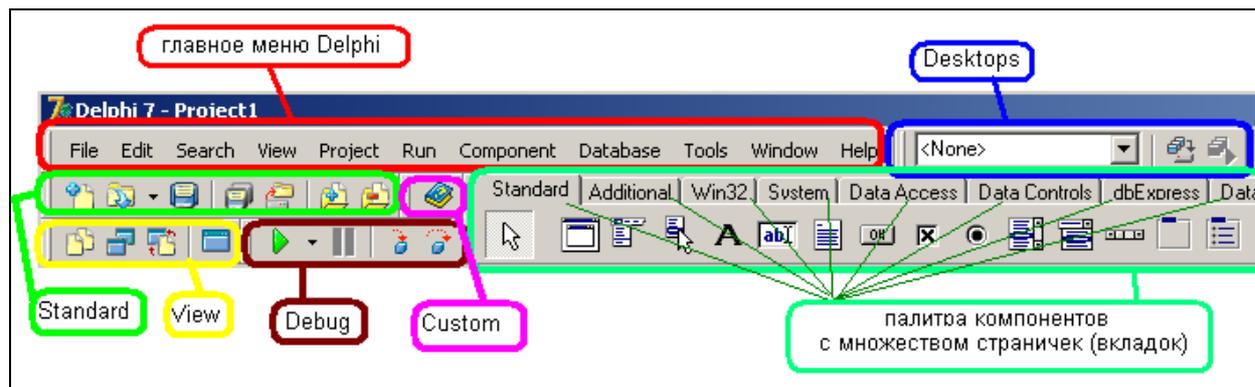


Рис.2.

Содержание и предназначение панелей инструментов, имеющих в начальном виде, следующее:

**Debug** - отладка. Позволяет запустить программу (Run), приостановить ее выполнение (Pause), а так же выполнять построчное выполнение программы;

**Standard** - стандартные. Служит для таких операций, как сохранение, создание, добавление и удаление файлов;

**View** - вид. Используется для быстрого нахождения форм и файлов проекта;

**Desktops** - рабочая среда. С помощью этих инструментов можно переключаться между различными настройками рабочей среды Delphi;

**Custom** - произвольная. Изначально содержит одну-единственную кнопку - для вызова справки;

**Component palette** - палитра компонентов. Содержит все доступные для разработки приложений компоненты.

*Палитра компонентов* - занимает правую часть главного окна и имеет закладки (странички), обеспечивающие быстрый поиск нужного компонента. Под *компонентом* понимается некий *функциональный элемент*, содержащий определенные *свойства* и размещаемый программистом в окне формы. С помощью компонентов создается каркас программы, во всяком случае - ее видимые на экране внешние проявления: *окна*, *кнопки*, *списки выбора* и т. д.

Все *компоненты* сгруппированы по *вкладкам*, число и состав которых несколько разнятся в зависимости от версии и варианта поставки. Так, в Delphi 7 Enterprise имеется 33 вкладки, содержащие компоненты, принадлежащие к той или иной группе VCL (VCL - это библиотека-хранилище компонентов) (табл.1).

Таблица 1. Страницы палитры компонент Delphi 7 Enterprise

| Страница        | Название                    | Описание  |
|-----------------|-----------------------------|---|
| Standard        | Стандартные                 | Основные элементы интерфейса приложений Windows (меню, кнопки, подписи и т.п.)          |
| Additional      | Дополнительные              | Набор улучшенных элементов управления, имеющихся в VCL                                  |
| Win32           | 32-разрядные Windows        | Элементы интерфейса приложений, характерные для Windows 95 и последующих версий этой ОС |
| System          | Системные                   | Элементы управления и доступа к системным функциям Windows (таймер, OLE, DDE)           |
| Data Access     | Доступ к данным             | Стандартный набор компонент для доступа к БД  |
| Data Controls   | Элементы данных             | Элементы пользовательского интерфейса для доступа к БД                                  |
| dbExpress       | dbExpress                   | Компоненты для доступа к БД при помощи SQL-драйвера dbExpress                           |
| WebServices     | WebServices                 | Компоненты для взаимодействия с удаленным web-сервером через SOAP                       |
| DataSnap        | DataSnap                    | Компоненты для взаимодействия с сервером через DCOM                                     |
| BDE             | Borland Database Engine     | Компоненты для доступа к БД посредством BDE (классический вариант для простых БД)       |
| ADO             | ADO                         | Компоненты для взаимодействия с БД через ADO  |
| InterBase       | InterBase                   | Компоненты для прямого взаимодействия с БД InterBase                                    |
| InterBase Admin | Администрирование InterBase | Компоненты для взаимодействия и управления сервером БД InterBase                        |
| InternetExpress | InternetExpress             | Компоненты для взаимодействия с данными через XML                                       |
| WebSnap         | WebSnap                     | Компоненты для работы с данными через различные протоколы Интернета                     |
| Internet        | Internet                    | Набор ActiveX-компонент для работы через Интернет                                       |
| Decision Cube   | Decision Cube               | Набор компонент для обработки информации в БД   |
| Dialogs         | Диалоги                     | Стандартные и расширенные диалоговые окна   |
| Win 3.1         | Windows 3.1                 | Компоненты пользовательского интерфейса, характерные для Windows 3.1                    |
| Samples         | Примеры                     | Несколько визуальных компонент, не являющихся официально поддерживаемыми                |
| ActiveX         | ActiveX                     | Несколько встраиваемых ActiveX-приложений   |

| Страница                                | Название          | Описание  |
|---|-------------------|---|
| Rave                                    | Rave Reports      | Набор компонент для построения отчетов  |
| Indy Clients                            | Клиенты Indy      | Набор компонент-клиентов для различных протоколов и служб Интернета   |
| Indy Servers                            | Серверы Indy      | Набор компонент-серверов для различных протоколов и служб Интернета   |
| Indy Intercepts                         | Обработчики Indy  | Набор компонент, позволяющих отлавливать сообщения от клиентов и серверов Indy  |
| Indy i/o handlers                       | Ввод-вывод Indy   | Компоненты для отслеживания активности соединений других компонент Indy   |
| Indy Misc                               | Утилиты Indy      | Набор вспомогательных компонент, полезных при разработке различных TCP-приложений   |
| COM+                                    | COM+              | Содержит компонент, позволяющий создать управляющий сервер COM+   |
| IW Standard, Data, Client Side, Control | IntraWeb          | Набор специальных кросс платформенных компонент для создания Web-приложений для любых Web-клиентов, включая КПК и смартфоны |
| Servers                                 | MS Office Servers | Набор ActiveX-компонент для взаимодействия с приложениями Microsoft Office  |

Суммарно Delphi включает в себя сотни компонент, мы выделим наиболее полезные для нас группы, а именно: *стандартные, дополнительные, 32-разрядные Windows, системные и диалоги*. Со временем Вы ознакомитесь так же с классическими компонентами для доступа к БД (Data Access и Data Controls).

Перейдем к детальному исследованию *главного меню Delphi*, которое состоит из 11 пунктов:

- **File** - файл. Операции с файлами, вроде создать, открыть, сохранить;
- **Edit** - правка. Операции редактирования, как стандартные для текстового процессора (отмена, копирование-вставка), так и специфические для редактирования разрабатываемых окон приложений (выравнивание, порядок со-здания и т.п.);
- **Search** - поиск. Различные варианты поиска и замены;
- **View** - вид. Переключение между различными окнами - как относящимися к IDE, так и к разрабатываемому приложению;
- **Project** - проект. Все операции по работе с проектом, как то добавление и удаление файлов, настройки, сборка и компиляция;
- **Run** – выполнить (запуск). Средства для отладки программ;
- **Component** - компоненты. Средства для работы с компонентами, включая настройку палитры компонент;
- **Database** - Данные. Некоторые средства для работы с БД;
- **Tools** - сервис. Настройка параметров IDE, а так же вызов вспомогательных программ (Image editor и др.);
- **Windows** - окно. Содержит список всех открытых в текущий момент окон и позволяет переключаться между ними (актуально, когда окон много и одни загораживают другие);
- **Help** - справка.

**ОКНО ДЕРЕВА ОБЪЕКТОВ (2)** предназначено для наглядного отображения связей между отдельными компонентами, размещенными на активной форме или в активном модуле данных. Щелчок по любому компоненту в этом окне активизирует соответствующий компонент в окне формы и отображает свойства этого компонента в окне *Инспектора объектов*. Двойной щелчок приводит к срабатыванию механизма Code Insight, который вставляет в окно кода заготовку для обработчика события OnClick.

**СОВМЕЩЕННОЕ ОКНО РЕДАКТОРА КОДА И ПРОВОДНИКА КОДА (3).** *Окно редактора кода* (или просто *окно кода*) предназначено для создания и редактирования текста программы. Этот текст составляется по специальным правилам и описывает алгоритм работы программы. Совокупность правил записи текста называется *языком программирования*. В системе Delphi используется язык программирования *Object Pascal*.

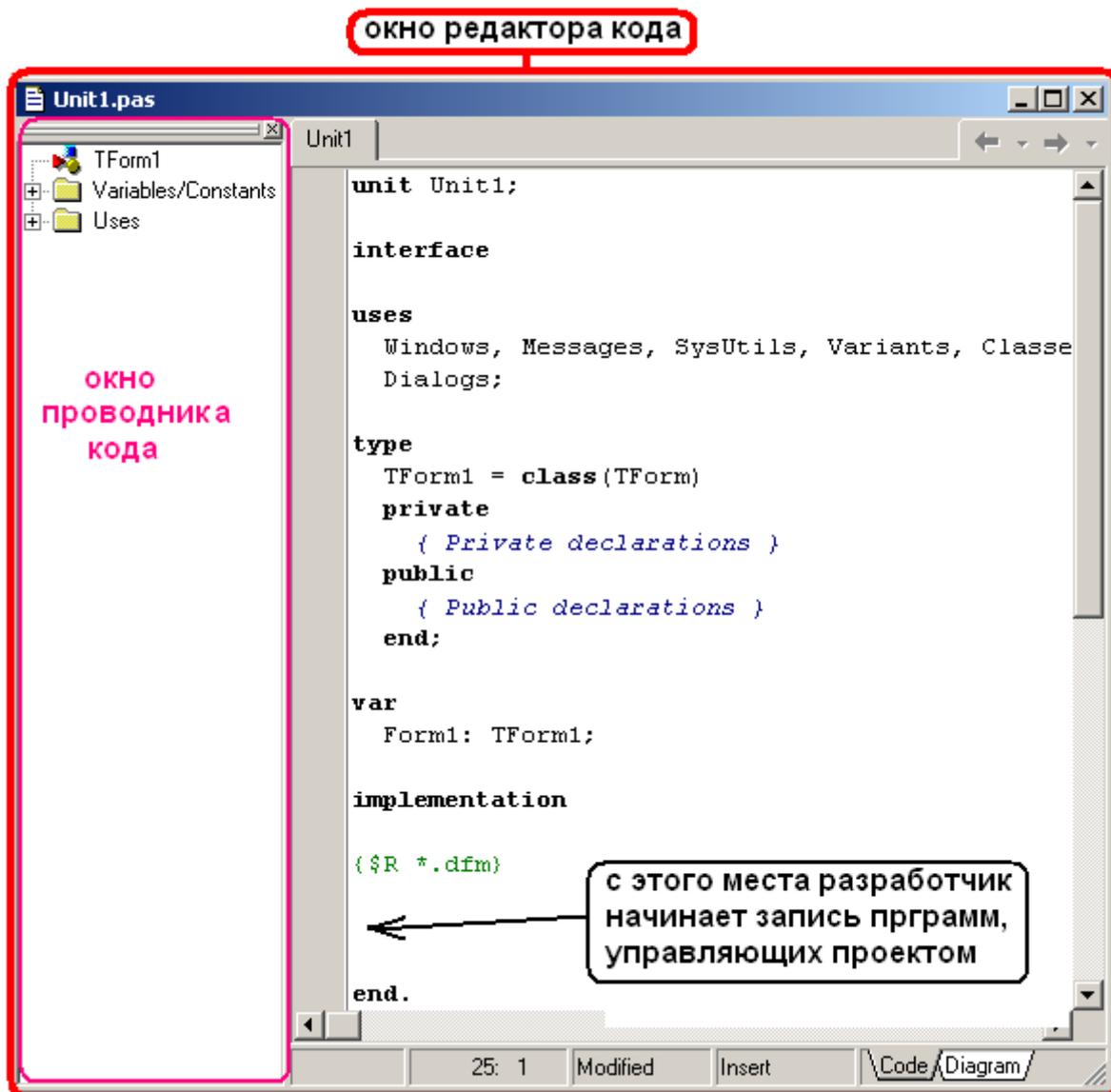


Рис.3.

Первоначально окно кода содержит минимальный исходный текст, обеспечивающий нормальное функционирование пустой формы в качестве полноценного Windows-окна. В ходе работы над проектом программист вносит в него необходимые дополнения, чтобы придать программе нужную функциональность.

Окно проводника кода пристыковано к левому краю окна редактора. При необходимости окно проводника кода можно закрыть. Вновь его установить можно через контекстное меню, вызванное на окне редактора кода → View Explorer.

**ОКНО КОНСТРУКТОРА ФОРМ (ОКНО ФОРМЫ) (4).** Окно формы представляет собой проект Windows-окна будущей программы. Вначале это окно пусто. Точнее, оно содержит стандартные для Windows интерфейсные элементы: кнопки вызова системного меню, максимизации, минимизации и закрытия окна, полосу заголовка и очерчивающую рамку. Вся рабочая область окна обычно заполнена точками координатной сетки, служащей для упорядочения размещаемых на форме компонентов (вы можете убрать эти точки, вызвав с помощью меню Tools → Environment options соответствующее окно настроек и убрав флажок в переключателе Display Grid на окне, связанном с закладкой Preferences). Значительную часть времени разработчик приложения (проекта) занят работой, напоминающей игру конструктором, он “достаёт” из палитры компонентов, как из коробки с деталями, нужный компонент и размещает его на “наборном поле” окна формы, постепенно заполняя форму интерфейсными элементами. Именно в этом процессе наполнения формы и заключается главная изюминка визуального программирования. Программист в любой момент времени контролирует содержание окна создаваемого приложения (проекта) и может внести в него необходимые изменения.

**ОКНО ИНСПЕКТОРА ОБЪЕКТОВ (5).** Любой размещаемый на форме компонент характеризуется некоторым набором параметров: положением, размером, цветом и т. д. Часть этих параметров, например, положение и размеры компонента, программист может изменять, манипулируя с компонентом в окне формы. Для изменения других параметров предназначено окно Инспектора объектов. Это окно содержит две страницы - Properties (Свойства) и Events (События).

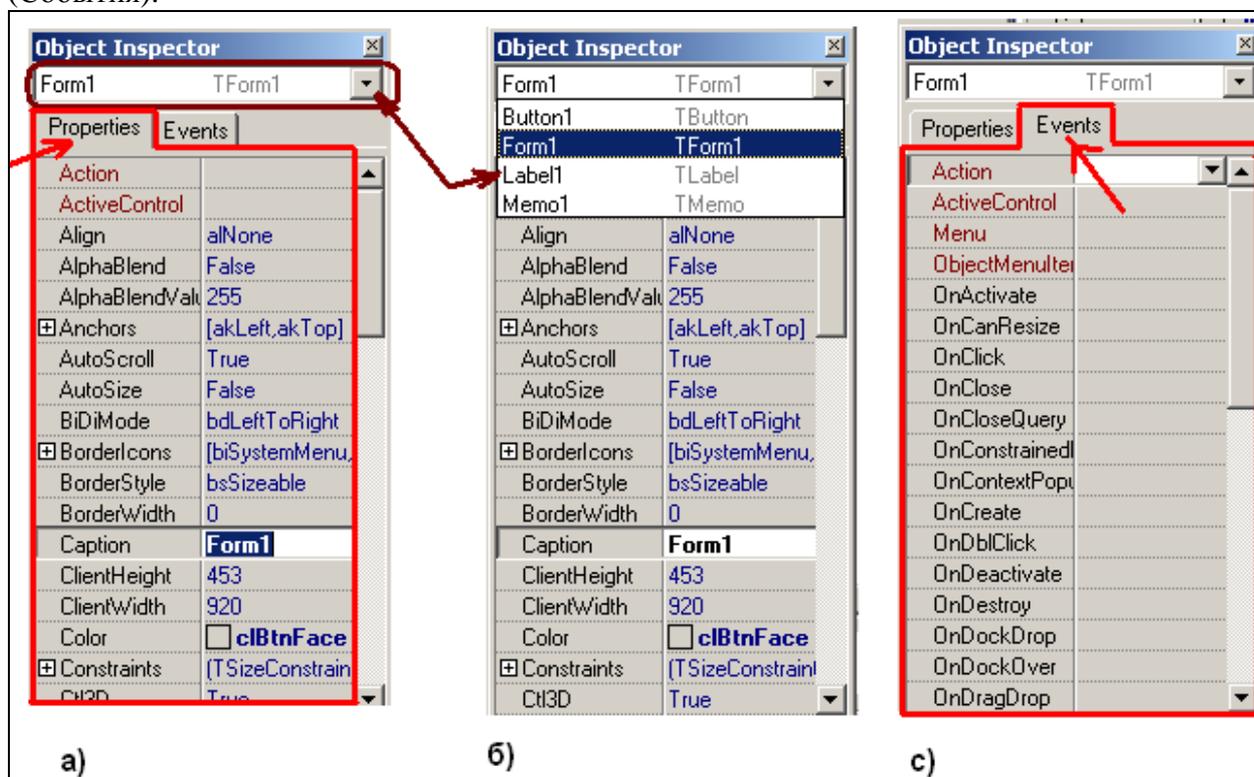


Рис.4.

В верхней части окна Инспектора объектов располагается раскрывающийся список всех помещенных на форму компонентов (рис.4.а) → б)). Поскольку форма сама по себе является

компонентом, ее имя также присутствует в этом списке. Раскрывающийся список содержит не только имена компонентов, но и их классы.

Страница Properties служит для установки нужных *свойств* компонента, страница Events позволяет определить *реакцию* компонента на то или иное событие. *Совокупность свойств* отображает видимую сторону компонента: положение относительно левого верхнего угла рабочей области формы, его размеры и цвет, шрифт и текст надписи на нем и т. п.; *совокупность событий* - его поведенческую сторону: будет ли компонент реагировать на щелчок мыши или на нажатие клавиш, как он будет вести себя в момент появления на экране или в момент изменения размеров окна и т. п.

Каждая страница окна *Инспектора объектов* представляет собой *двухколоночную таблицу*, левая колонка которой содержит *название свойства* или *события*, а правая - конкретное *значение* свойства или *имя подпрограммы*, обрабатывающей соответствующее событие.

Строки таблицы выбираются щелчком мыши и могут отображать простые или сложные свойства. К простым относятся *свойства*, определяемые единственным значением - числом, строкой символов, значением True (Истина) или False (Ложь) и т. п. Например, свойство Caption (Заголовок) представляется строкой символов, свойства Height (Высота) и Width (Ширина) - числами, свойство Enabled (Доступность) - значениями True или False. Сложные свойства определяются *совокупностью значений*. Слева от имени таких свойств указывается значок "+", а щелчок мышью по этому символу приводит к *раскрытию списка* составляющих сложного свойства. Чтобы закрыть раскрытый список, нужно щелкнуть по значку "-" сложного свойства.

Если вы случайно или намеренно сделаете окно Инспектора объектов невидимым, нажмите F11 или выберите опцию View → Object Inspector, чтобы оно вновь появилось на экране.

Выполнить:

1. Открыть среду Delphi: Пуск → Программы → BorlandDelphi7 → Delphi7. Если появилось *Окно связи с Inprise (Borland)*, закройте его.
2. Закройте следующие окна: *окно дерева объектов (Object TreeView)* и *окно инспектора объектов (Object Inspector)*. Вновь установите эти окна из главного меню пункт View.
3. Сверните (но не закрывайте) среду Delphi и на диске D создайте Вашу личную папку, а в ней еще одну папку → «Занятие1». Вновь разверните среду Delphi и выполните сохранение еще пустого проекта в папку «Занятие1»: File → SaveProjectAs... Выполнится сохранение двух файлов: Unit1 и Project1.
4. Закройте окна: конструктора форм – саму форму Form1 и окно кода – Unit1 (если всплывет окно, с сообщением о сохранении изменений в Unit1, можете нажать Yes или No, рис.5.)

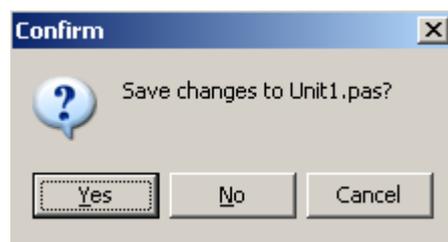


Рис.5

5. Чтобы вновь восстановить эти окна (Форму и окно кода), необходимо вызвать Менеджер проекта из главного меню View → ProjectManager и выполнить действия в соответствии с рис.6.

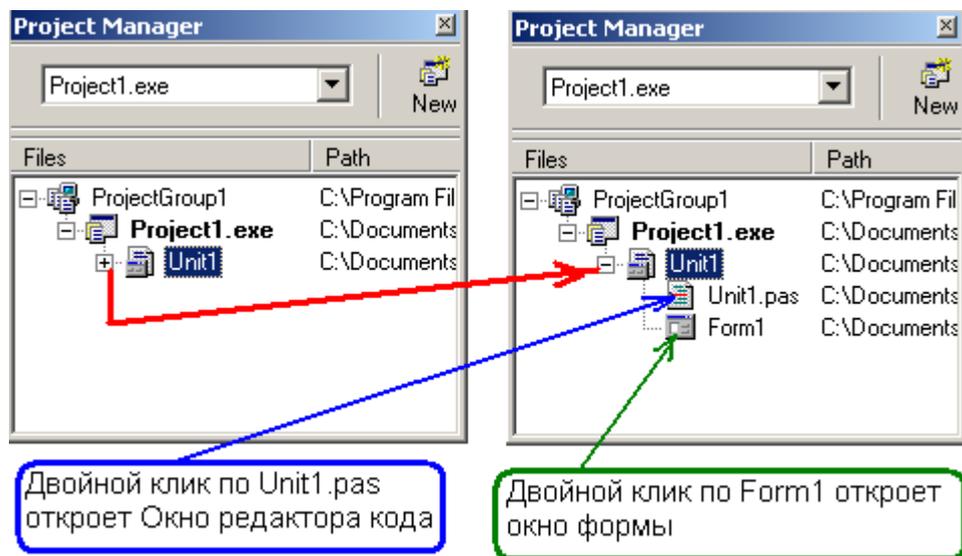


Рис.6

6. Закройте все панели инструментов: **Debug** , **View** , **Desktops** , **Custom** , **Component palette** и вновь установите их.
7. Ознакомьтесь с составом закладок *Standard* и *Additional* палитры компонентов и нанесите на Форму следующие компоненты с этих страничек: **Label1**, **Button1**, **Panel1**, **SpeedButton1**, **Image1**, **ScrollBar1**.
8. Запустите созданное Вами приложение следующим образом: пункт главного меню **Run** → **Run** (или нажатие клавиши клавиатуры **F9**).
9. Остановите запущенное приложение следующим образом: пункт главного меню **Run** → **ProgramReset** (не путайте с командой **ProgramPause**).
10. Досохраните Ваш проект следующим образом: пункт главного меню **File** → **Save All**.
11. Закройте среду Delphi нажав на кнопку закрытия в *главном окне*  и вновь запустите Delphi. Перед Вами новое пустое приложение. Откройте только что созданный Вами проект следующим образом: пункт главного меню **File** → **OpenProject**.
12. Просмотрите через Инспектор объектов состав компонентов в Вашем проекте (как показано на рис.4: а) и б) ).

При защите лабораторной работы №1 студент должен знать ответы на следующие вопросы:

1. Основные окна среды Delphi.
2. Назначение окна *Инспектор объектов*, его основные части. Уметь закрыть и вновь установить *Инспектор объектов*.
3. Назначение *окна редактора кода*, его основные части. Уметь закрыть и вновь установить *окно редактора кода*.
4. Назначение *окна формы*. Уметь закрыть и вновь установить *окно формы*.
5. Назначение главного окна среды Delphi, его основные части. Уметь закрыть и вновь установить на главное окно среды Delphi все панели инструментов.
6. Назначение палитры компонентов.
7. Название компонентов, находящихся на страницах *Standard* и *Additional* палитры компонентов.
8. Где находятся странички Properties и Events.
9. Перечень всех пунктов *главного меню Delphi* и их назначение.
10. Для чего нужен Менеджер проекта и как вызвать его окно.

11. Что такое окно связи и как (когда) оно появляется.
12. Как (какой командой) сохраняется еще пустое приложение.
13. Как (какой командой) сохраняется разрабатываемое приложение.
14. Как открыть ранее созданный проект.
15. Содержит ли окно Редактора программного кода какие либо записи, если приложение еще не разрабатывалось (т.е. среда *Delphi* только запущена и перед Вами новое, пустое приложение).

В тетради по практическим занятиям по дисциплине «ППЗ» должны быть законспектированы краткие ответы на контрольные вопросы к лабораторной работе.

## Лабораторная работа №2

### Тема: Проекты в Delphi

Начиная с этой лабораторной работы, мы будем учиться создавать приложения. Приложения помогают реализовать решение определенной задачи с помощью компьютера. Итак:

- приложение - это достаточно сложная вещь, которая *создается из различных частей*;  
- *каждая часть приложения размещена(хранится) в отдельном файле* и выполняет строго определенные функции; совокупность файлов, необходимых для создания приложения, называется проектом;

- компилятор последовательно обрабатывает файлы проекта и строит из них выполняемый (исполняемый) файл.

Итак, проект - это группа файлов, представляющих исходные данные (прежде всего, программный код) для приложения. Одни из этих файлов создаются во время разработки приложения (собственно программный код, включая файл проекта, и представленные в виде кода формы), другие же создаются автоматически при запуске программы. Все файлы проекта подразделяются на следующие типы (ниже даны расширения файлов):

- .dpr - собственно файл проекта;
- .pas - модули приложения, содержащие код на Object Pascal;
- .dfm - модули приложения, содержащие информацию об окнах приложения;
- .res - файлы со встраиваемыми ресурсами приложения (например, иконками);
- .obj - файлы, содержащие готовый к компиляции объектный код;
- .cfg, .dof, .dsk - служебные файлы Delphi.

Перед тем, как открыть среду Delphi и начать разработку (создание) нового проекта, Вы должны создать папку, в которой будет храниться Ваш проект. Напомним, что первое сохранение еще пустого проекта выполняется сразу после открытия среды Delphi, командой File → SaveProjectAs. Пожалуйста, сохраняйте проекты в СВОЮ папку, а не где придется!!!

После сохранения проекта в Вашей папке образуется сразу шесть файлов, а после компиляции проекта их станет восемь.

а) так отображены файлы проекта в Вашей папке сразу после сохранения проекта (File→SaveProjectAs)

| Имя          | Тип                 |
|--------------|---------------------|
| Unit1        | Delphi Form         |
| Project1     | Delphi Project      |
| Unit1        | Delphi Source File  |
| Project1.dof | Файл "DOF"          |
| Project1     | Файл "RES"          |
| Project1     | Файл конфигураци... |

здесь добавлены расширения к именам файлов

| Имя          | Тип                 |
|--------------|---------------------|
| Unit1.dfm    | Delphi Form         |
| Project1.dpr | Delphi Project      |
| Unit1.pas    | Delphi Source File  |
| Project1.dof | Файл "DOF"          |
| Project1.res | Файл "RES"          |
| Project1.cfg | Файл конфигураци... |

б) и после компиляции приложения (добавляется еще два файла)

| Имя          | Тип                 |
|--------------|---------------------|
| Unit1        | Delphi Form         |
| Project1     | Delphi Project      |
| Unit1        | Delphi Source File  |
| Project1.dof | Файл "DOF"          |
| Project1     | Файл "RES"          |
| Project1     | Файл конфигураци... |
| Project1     | Приложение          |
| Unit1.dcu    | Файл "DCU"          |

| Имя          | Тип                 |
|--------------|---------------------|
| Unit1.dfm    | Delphi Form         |
| Project1.dpr | Delphi Project      |
| Unit1.pas    | Delphi Source File  |
| Project1.dof | Файл "DOF"          |
| Project1.res | Файл "RES"          |
| Project1.cfg | Файл конфигураци... |
| Project1.exe | Приложение          |
| Unit1.dcu    | Файл "DCU"          |

*Компилирование программы* - это перевод исходной программы в эквивалентную ей результирующую программу на языке машинных команд. Другими словами, компилятор превращает код программы на "человеческом" языке в объектный код понятный компьютеру.

Проект Delphi-приложения состоит из трех основных типов файлов: самого файла проекта (dpr) и файлов модулей pas и dfm. При этом для каждого модуля окна (dfm) имеется собственный программный модуль (pas).

### Описание файлов проекта

Главной частью приложения является файл проекта (.dpr) с которого начинается выполнение программы и который обеспечивает инициализацию других модулей.

Модуль – автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно некоторые исполняемые операторы иницилирующей части.

**1. Главный файл проекта** - текстовый файл с расширением **dpr**, он представляет собой главный программный файл на языке Object Pascal, который подключает с помощью оператора **uses** все файлы модулей, входящих в проект. Для каждого проекта существует только один dpr-файл. Файл проекта подключает все используемые программные модули и содержит операторы для запуска приложения. Этот файл среда Delphi создает и контролирует сама.

Чтобы увидеть содержимое **dpr**-файла нашего приложения, выберите в главном меню пункт View→Units... → далее выбрать строку Project1. В окне «Редакторе кода» появится новая страница со следующим текстом (рис.7):

```

program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

($R *.res)

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.

```

Рис.7.

Подключение модуля Forms обязательно для всех программ, так как в нем содержится определение *объекта* Application. Этот объект лежит в основе любого Delphi-приложения и доступен на протяжении всей его работы.

Подключаемый следом модуль Unit1 содержит определение *формы приложения*. Название формы приводится в фигурных скобках. Директива **in** указывает на то, что модуль является необходимой частью проекта и существует в виде исходного текста на языке Object Pascal.

Директива **{SR \*.res}** подключает к результирующему exe-файлу так называемые ресурсы в данном случае - значок приложения. Этот значок будет виден на «Панели задач Windows».

Дальше следует главный программный блок, содержащий вызовы трех методов объекта Application:

- вызов метода `Initialization` подготавливает приложение к работе,
- метод `CreateForm` загружает и инициализирует форму `Form1`,
- метод `Run` активизирует форму и начинает выполнение приложения. Фактически

время работы метода Run – это время работы приложения. Выход из метода Run происходит тогда, когда пользователь закрывает главную форму приложения; в результате приложение завершается.

**Внимание!!!** Никогда не изменяйте **dpr** -файл вручную. Оставьте эту работу для Delphi.

Продолжим описание содержания главного файла проекта.

Итак, *файл проекта* представляет собой программу, которая автоматически создается Delphi и содержит лишь несколько строк.

В окне кода **жирным** шрифтом выделяются так называемые **зарезервированные** слова, а *курсивом* – **комментарии**. Текст программы начинается зарезервированным словом **program** и заканчивается словом **end** с точкой за ним.

**ВНИМАНИЕ!!!**, сочетание **end** со следующей за ней *точкой* называется **терминатором программной единицы**: как только в тексте программы встретится такой **терминатор**, компилятор **прекращает** анализ программы и **игнорирует оставшуюся часть текста**.

*Зарезервированные слова* играют важную роль в Object Pascal, придавая программе в целом свойство текста, написанного на почти естественном английском языке. Каждое зарезервированное слово (а их в Object Pascal несколько десятков) несет в себе условное сообщение для компилятора, который анализирует текст программы так же, как читаем его и мы: слева направо и сверху вниз.

*Комментарии*, наоборот, ничего не значат для компилятора, и он их игнорирует. Комментарии важны для программиста, который с их помощью поясняет те или иные места

программы. Наличие комментариев в тексте программы делает ее понятнее и позволяет легко вспомнить особенности реализации программы, которую вы написали несколько лет назад. В Object Pascal комментарием считается любая последовательность символов, заключенная в фигурные скобки. В приведенном выше тексте таких комментариев два, но строка

```
{ $R *.RES }
```

на самом деле **не является комментарием**. Этот специальным образом написанный фрагмент кода называется *директивой* компилятора (в нашем случае - указание компилятору на необходимость подключения к программе так называемого файла ресурсов). Директивы начинаются символом \$, который стоит сразу за открывающей фигурной скобкой.

В качестве ограничителей комментария могут также использоваться пары символов (\*, \*) и //. Скобки (\*...\*) используются подобно фигурным скобкам т. е. комментарием считается находящийся в них фрагмент текста, а символы // указывают компилятору, что комментарий располагается за ними и продолжается до конца текущей строки:

```
{ Это комментарий }
```

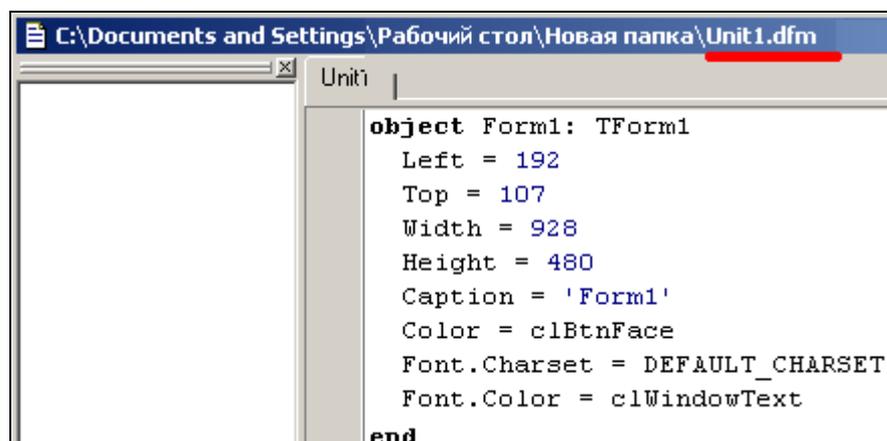
```
(* Это тоже комментарий *)
```

```
// Все символы до конца этой строки составляют комментарий
```

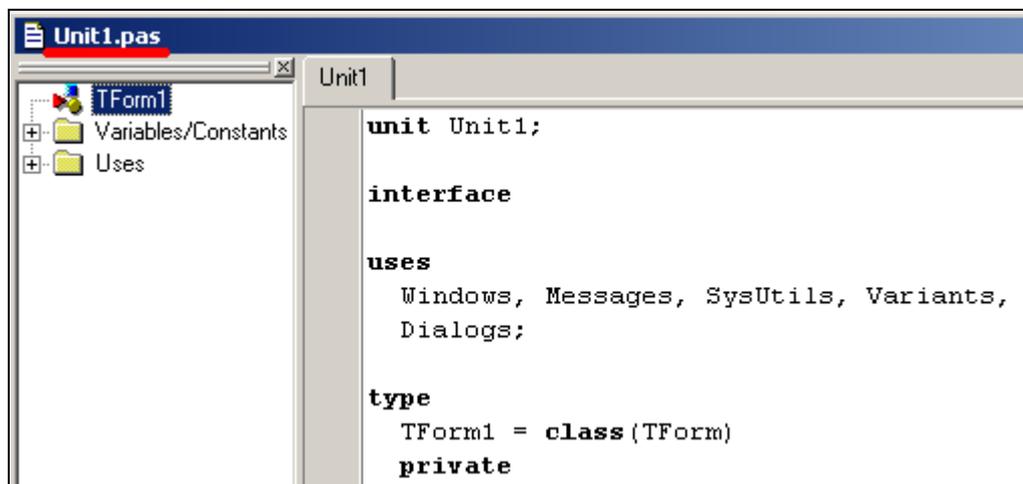
Слово **program** со следующим за ним именем программы и точкой с запятой образуют *заголовок программы*. За заголовком следует *раздел описаний*, в котором программист (или Delphi) описывает используемые в программе *идентификаторы*. Идентификаторы обозначают элементы программы, такие как типы, переменные, процедуры, функции. Здесь же с помощью предложения, которое начинается зарезервированным словом **uses** (использовать) программист сообщает компилятору о тех фрагментах программы (модулях), которые необходимо рассматривать как неотъемлемые составные части программы и которые располагаются в других файлах.

Собственно *тело* программы начинается со слова **begin** (начать) и ограничивается терминатором **end** с точкой. Тело состоит из нескольких операторов языка Object Pascal. В каждом *операторе* реализуется некоторое действие - изменение значения переменной, анализ результата вычисления, обращение к подпрограмме и т. п. В *теле* нашей программы - три исполняемых оператора: Initialize, CreateForm, Run.

**2. Файлы описания форм** - двоичные файлы с расширением **dfm**, описывающие формы с компонентами. В этих файлах запоминаются начальные значения свойств, установленные вами в «Инспекторе Объектов».



**3. Файлы программных модулей** - текстовые файлы с расширением **pas**, содержащие исходные коды форм на языке Object Pascal. В этих файлах вы пишете методы обработки событий, генерируемых формами и компонентами.



4. Файл параметров проекта(.dfo). В этом файле хранятся установки параметров проекта

5. Файл ресурсов(.res). Этот бинарный файл содержит используемую проектом пиктограмму и прочие ресурсы.

6. Файлы резервных копий (.~dp, .~df, .~pa). Это соответственно файлы резервных копий для файлов проекта, формы и модуля (.~dp → dpr, .~df → dfm, .~pa → pas). Если что-то безнадежно испорчено в проекте, можно соответственно изменить расширения этих файлов и таким образом вернуться к предыдущему не испорченному варианту.

7. Файл конфигурации окон (.dsk). Файл хранит конфигурацию всех окон среды разработки.

8. Исполняемый файл (.exe). Это исполняемый файл вашего приложения. Он является автономным исполняемым файлом, для которого больше ничего не требуется, если только вы не используете библиотеки, содержащиеся в DLL, OCX и т.д., а также если вы не используете поддержку пакетов времени выполнения.

9. Объектный файл модуля (.dcu). Это откомпилированный файл модуля (.pas), который компонуется в окончательный исполняемый файл.

Кроме этих основных файлов проект Delphi может содержать: динамически присоединяемую библиотеку \*.dll; файлы справки \*.hlp; файлы изображений \*.wmf, \*.bmp, \*.ico; анимационные файлы \*.avi; файл группы файлов \*.brg, который создается средой, в случае если идет работа с группой файлов.

Для перемещения Delphi-проекта необходимы только файлы \*.pas, \*.dpr, \*.dfm, \*.res. Остальные файлы создаются автоматически.

Выполнить.

1. В Вашей личной папке создайте папку Лабораторная2.

2. Запустите среду Delphi и сохраните в папку Лабораторная2 ещё пустой проект (File → SaveProjectAs...).

3. Сверните среду Delphi, откройте через «Проводник» или «Мой компьютер» папку Лабораторная2 и просмотрите образовавшиеся в ней файлы (их должно быть шесть): файл Unit1.pas - программный код для формы, Unit1.dfm - описание формы, Project1.dpr - сам проект, Project1.res - файл ресурсов проекта, служебные файлы Delphi, в которых хранится дополнительная информация о проекте и настройках рабочей среды для него - файлы Project1.cfg, Project1.dof.

4. Вновь «разверните» среду Delphi и запустите еще пустой проект одним из следующих способов:

- или нажмите клавишу F9;

- или меню Run → Run;

- или на панели инструментов Debug кнопку запуска (см. рисунок ниже):



5. Перед Вами будет обычная форма, только без перфорированной сетки. Также на экране будет отсутствовать Инспектор объектов.

6. Следующее Ваше действие – это остановка запущенного приложения. Выполните: пункт главного меню **Run → ProgramReset** (не путайте с командой **ProgramPause**). Обратите внимание, на экране вновь появился Инспектор объектов.

7. Сверните среду Delphi, и просмотрите содержимое папки Лабораторная2. В ней должно быть восемь файлов. После команды запуска приложения (Run → Run или другим способом) машина компилирует исполняемый файл готового приложения (exe-файл) и создает файл Unit1.dcu - подготовленный к компиляции модуль – эти два файла и добавились в папку с Вашим проектом.

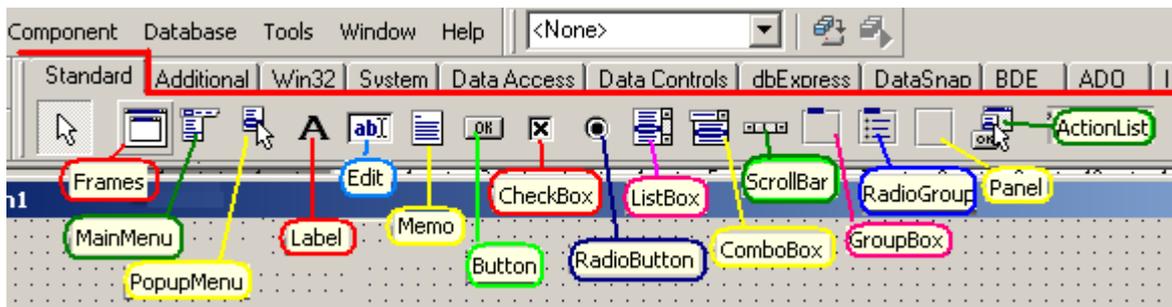
При защите «Лабораторной работы №2» студент должен знать ответы на следующие вопросы:

1. Что такое проект.
2. Из каких файлов состоит проект.
3. Что такое компилирование программы.
4. Назовите три основных файла проекта.
- 5.. Функциональное назначение *главного файла проекта*.
6. Функциональное назначение *файла описания формы*.
7. Функциональное назначение *файла программного модуля*.
8. Что такое exe-файл и когда он образуется
9. Что такое *резервированные* слова, приведите пример.
10. Назначение *комментариев* в программе, приведите пример написания комментария.
11. Что такое терминатор программной единицы. Его функции.
12. Что такое директивы компилятора, как они обозначаются в программе.
13. При перемещении проекта из одной папки в другую какие файлы обязательно переносятся, а какие создаются автоматически после очередного запуска приложения.
14. Какой командой выполняется первичное сохранение ещё пустого проекта.
15. Как запустить приложение.
16. Как остановить запущенное приложение.

В тетради по практическим занятиям по дисциплине «ППЗ» должны быть законспектированы краткие ответы на контрольные вопросы к лабораторной работе.

### Лабораторная работа №3

Тема: Компоненты, расположенные на страничке *Палитры компонентов Standard*.



Из названия вкладки следует, что компоненты, представленные на ней, являются стандартными, системными.

**Frames** - позволяет разместить на форме так называемый "фрейм". Фрейм представляет из себя другое окно. Чтобы создать окно-фрейм, следует выбрать пункт меню **File** → **New** → **Frame**, либо выделить значок **Frame** в окне **File** → **New** → **Other** на вкладке **New**. До тех пор, пока в приложении не будет ни одного *фрейма*, использовать данный объект не удастся. Фреймы удобны в том случае, когда какие-либо настройки запрашиваются во время работы программы в виде отдельной формы, а также, например, на одной из вкладок основной формы.

**MainMenu** – *главное* (основное) *меню* окна. Связать его с формой можно через свойство *Menu* формы. При двойном щелчке по значку *MenuMenu* появляется дизайнер меню, в котором можно добавлять и удалять пункты. В свойстве *Caption* задаётся название (текст) пункта меню. Чтобы создать черту-разделитель, следует в свойство *Caption* прописать знак "минус" ("-", без кавычек).

**PopupMenu** - *контекстное меню* (вызывается правой кнопкой мыши). Его можно привязать ко многим компонентам (как правило, это делается через свойство *PopupMenu* у компонента, которому ассоциируется это меню). Одно и то же меню может быть привязано к нескольким компонентам.

**Label** - *текстовая метка (надпись)* на форме. Используется для отображения любого текста в окне. Текст задаётся в свойстве *Caption*. Свойство *Font* позволяет настроить шрифт текста.

**Edit** - *поле ввода*. Используется для ввода любых данных (текста, числа и т.д.), представленных одной строкой. Свойство *ReadOnly* позволяет запретить редактирование текста в поле. Текст хранится свойством *Text*. Свойство *MaxLength* позволяет задать максимальное число символов, которое может быть введено в поле. Значение 0 означает, что ограничение не установлено.

**Memo** - *многострочный Edit*. Используется для ввода больших объёмов текста. Свойство *ScrollBars* позволяет указать полосы прокрутки, которые будут отображаться у поля при недостатке места для всего текста:

*ssNone* - без полос прокрутки;

*ssHorizontal* - горизонтальная полоса прокрутки;  
*ssVertical* - вертикальная;  
*ssBoth* - обе: и горизонтальная, и вертикальная.

**Button** - кнопка, обыкновенная, нажатие на которую вызывает запрограммированное действие. Свойство *Caption* - текст на кнопке.

**CheckBox** - флажок-переключатель. Состояние флажка хранится в свойстве *Style*:  
cbUnchecked - не отмечен;  
cbChecked - отмечен;  
cbGrayed - затемнён.

Само состояние флажка следует изменять с помощью свойства *Checked*. Таких флажков может быть несколько и каждый может быть отмечен независимо от остальных.

**RadioButton** - радио-переключатель (альтернативный переключатель). Аналогичен **CheckBox**, но при наличии нескольких переключателей может быть выбран только один из них. Свойство *Checked* позволяет изменить состояние переключателя.

**ListBox** - список текстовых строк. Используется для выбора одного из вариантов. Строки задаются в свойстве *Items*. Чтобы изменить исходные значения, следует щёлкнуть по кнопке с тремя точкам (...) в строке свойства *Items* и в открывшемся редакторе ввести требуемые значения. Свойство *MultiSelect* позволяет *включить/выключить* одновременный выбор нескольких строк из списка.

**ComboBox** - ещё один список для выбора, но выпадающий - на экране видна всего одна строка, а сам список появляется при нажатии на кнопку со стрелкой. Установив свойство *Style* в *csDownList* можно запретить ввод произвольного значения в *ComboBox*. Текст выбранной строки доступен в свойстве *Text*.

**ScrollBar** - полоса прокрутки (бегунок). Свойство *Kind* определяет ориентацию полосы на форме: *sbHorizontal* - горизонтально, *sbVertical* - вертикально. Свойство *Max* - определяет максимальное (числовое) значение, *Min* - минимальное значение.

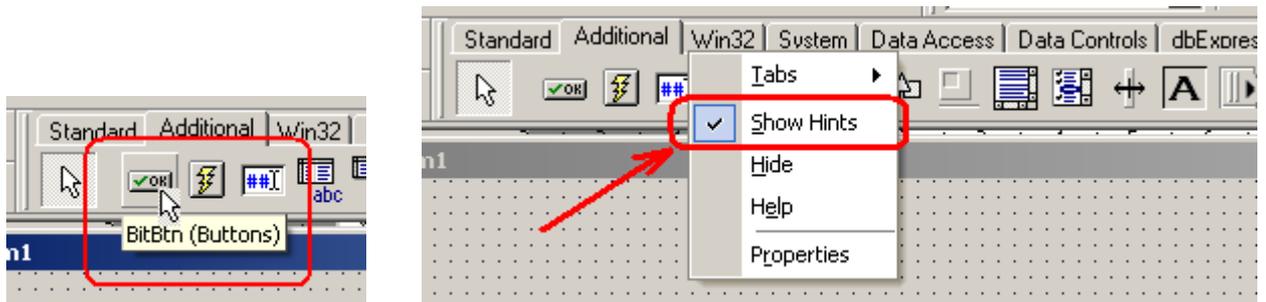
**GroupBox** - "контейнер" для компонент. Используется для объединения элементов в отдельные области. При перемещении *GroupBox*-а все компоненты, расположенные на нём, также перемещаются. Для размещения компонент в этом контейнере следует после выбора их на Палитре компонент щёлкнуть по самому контейнеру (а не по форме, как обычно). У *GroupBox* можно задать текст заголовка - свойство *Caption*.

**RadioGroup** - группа из нескольких *RadioButton*-ов. Тексты строк задаются в свойстве *Items*. Номер выбранной строки - свойство *ItemIndex* (нумерация строк начинается с нуля!).

**Panel** - практически такой же контейнер, что и *GroupBox*, однако без заголовка и с возможностью отключения рельефных границ. Обычно применяется при создании панелей инструментов.

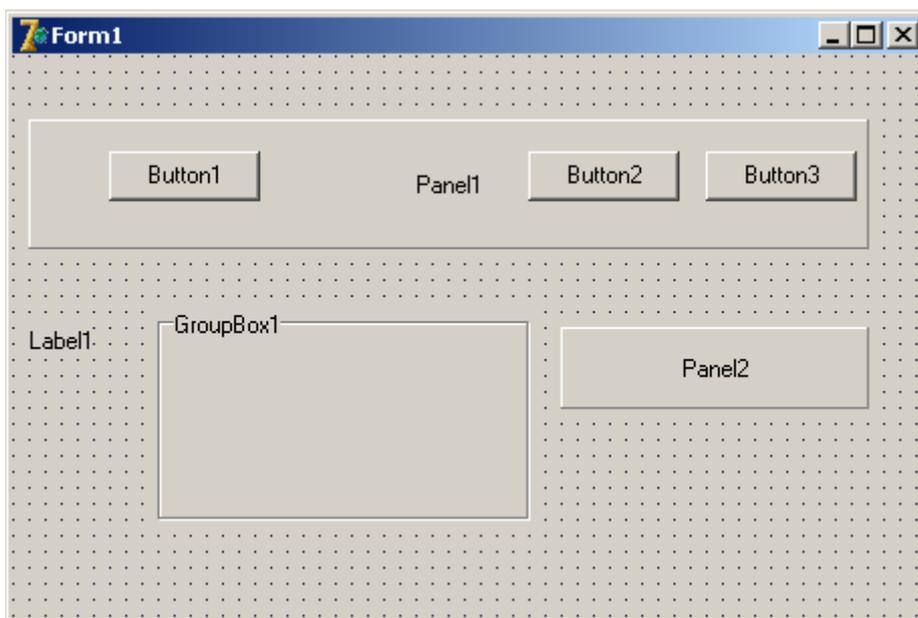
**ActionList** - позволяет управлять действиями (Actions), которые привязываются к пунктам меню, кнопкам и т.д.

**ПРИМЕЧАНИЕ.** Чтобы при наведении мышки на компоненту всплывала надпись (всплывающая подсказка) с названием компонента, должен быть выставлен флажок в пункте ShowHints в контекстном меню, вызываемом на *Палитре компонентов*.



**Выполнить.**

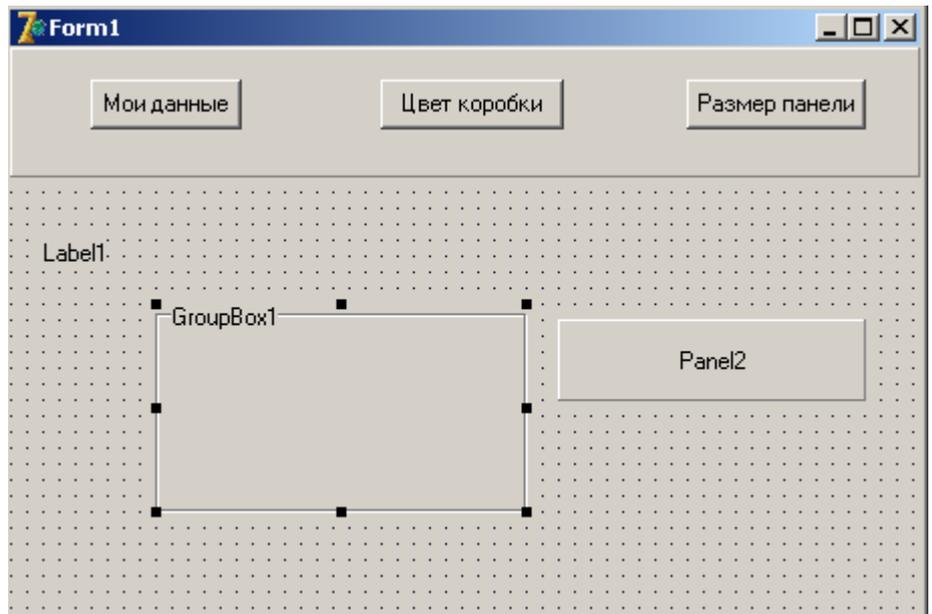
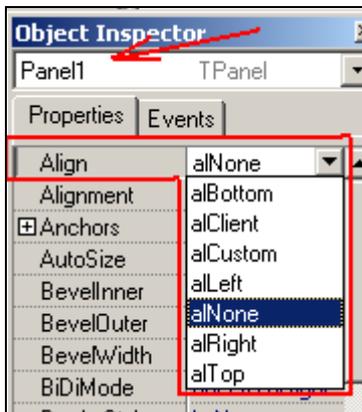
1. Запустите среду Delphi и *сохраните* проект в папку Лаб\_3-1 (создайте ее в процессе сохранения проекта в Вашей личной папке).
2. Нанесите на форму следующие компоненты:



3. Измените **надписи** на кнопках через *Инспектор объектов* (свойство **Caption**):
  - **Button1** → Мои данные
  - **Button2** → Цвет коробки
  - **Button3** → Размер панели.

4. Изменим *месторасположения* компонента **Panel1** на форме **Form1** – «прилепим» панель к верхнему краю формы. Для этого установите свойству **Align** значение *alTop*. Самостоятельно попробуйте поочерёдно применить все значения свойства **Align** и посмотрите, как меняется местоположение объекта.

Не забудьте снова установить свойству **Align** значение *alTop*!!!



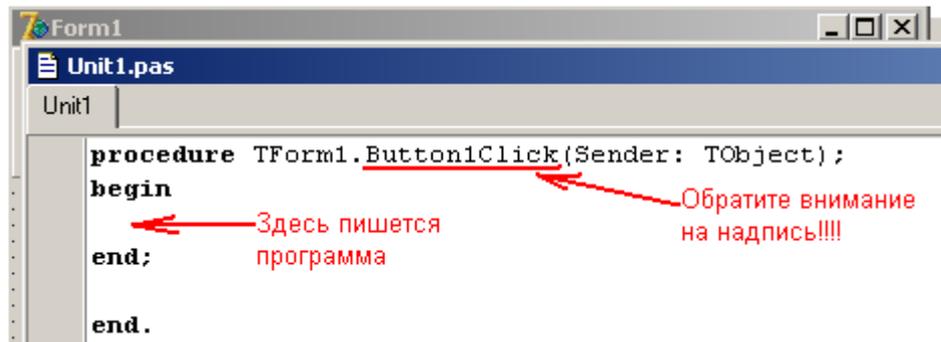
### 5. Изменение свойств компонента Label1:

- в инспекторе объектов установите значение свойства *WordWrap* → true – это позволит разместить надпись на объекте в несколько строк.

Далее переходим к программированию, будем программно изменять значения *свойств* компонентов.

Нам необходимо, чтобы нажатие на кнопку «Мои данные» выводило в компонент **Label1** надпись: «студент группы ДИ-12, Иванов Иван Иванович». Это значит, что мы программным путем меняем значение свойства *Caption* у компонента **Label1**. Для этого:

- выполните двойной клик мышкой по компоненту **Button1** (кнопка с надписью «Мои данные»);
- откроется окно *программного кода* со следующей заготовкой:



- между ключевыми словами **begin** и **end** пропишите следующий программный код:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Caption := 'Студент группы ДИ-12  ИВАНОВ ИВАН ИВАНОВИЧ';
end;
```

Обратите внимание на синтаксис программы:

**Имя объекта. Изменяемое свойство := новое значение свойства**

- запустите приложение (F9) и нажмите кнопку «Мои данные» - в компоненте **Label1** появилась новая надпись, расположенная в несколько строк;
- **ОСТАНОВИТЕ ЗАПУЩЕННОЕ ПРИЛОЖЕНИЕ!!!**, и в инспекторе объектов для компонента **Label1** измените значение свойства WordWrap → false. Вновь запустите приложение (F9) и нажмите кнопку «Мои данные» - размещение надписи в компоненте **Label1** изменилось. Вновь **ОСТАНОВИТЕ ЗАПУЩЕННОЕ ПРИЛОЖЕНИЕ**.

Во избежание непредвиденных ситуаций при создании проектов, связанных с самопроизвольным закрытием среды Delphi рекомендуется после каждого запуска приложения и его остановки **ДОСОХРАНЯТЬ** создаваемый Вами проект: меню File → Save All.

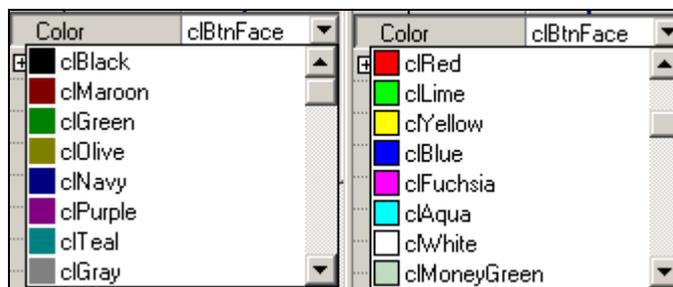
**6. Изменение свойств компонента GroupBox1.** Нам необходимо, чтобы при нажатии на кнопку «Цвет коробки» компонент **GroupBox1** заливался красным цветом.

- выполните двойной клик мышкой по компоненту **GroupBox1** (кнопка с надписью «Цвет коробки»);
- откроется окно *программного кода*, заполните его как показано ниже :

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    GroupBox1.Color := clRed;
end;
```

- запустите приложение (F9) и нажмите кнопку «Цвет коробки», **ОСТАНОВИТЕ ЗАПУЩЕННОЕ ПРИЛОЖЕНИЕ**.

При изменении цветов объектов используйте следующие значения:



**7. Изменение свойств компонента Panel2.** Нажатие на кнопку «Размер панели» должно увеличивать *длину* (свойство **Width** в инспекторе объектов) и *высоту* (**Height**) компонента **Panel2** на произвольное значение *пикселей* (например на 50).

- выполните двойной клик мышкой по компоненту **Panel2** (кнопка с надписью «Размер панели»);
- откроется окно *программного кода*, заполните его как показано ниже :

```
procedure TForm1.Button3Click(Sender: TObject);
begin
    Panel2.Width := Panel2.Width +50;
    Panel2.Height := Panel2.Height +50;
end;
```

- запустите приложение (F9) и нажмите кнопку «Размер панели», **ОСТАНОВИТЕ ЗАПУЩЕННОЕ ПРИЛОЖЕНИЕ**.

Разберитесь **САМОСТОЯТЕЛЬНО** в предоставленном программном коде, этот вопрос будет включен в контрольный опрос при защите лабораторной работы!!!

8. Закройте созданное ( и сохраненное!!!) Вами приложение File → Close All. Создайте новое приложение File → New → Application и сохраните его в папку Лаб\_3-2 (создайте ее в процессе сохранения проекта в Вашей личной папке). При сохранении *файла проекта* компьютер может дать ему *имя* Project2, а не Project1 (это особенности среды программирования, так как только что в ней уже разрабатывался Project1, а сама среда не закрывалась). Вы сами вручную исправьте 2 на 1, т.е. Project2 → Project1.

Измените **ИМЯ** формы в инспекторе объектов через свойство **Name** → MyForm (**Form1** сотрите, а **MyForm** напишите). Научитесь не путать *заголовок* формы (значение свойства **Caption** ) и её *имя* (свойство **Name**). Свойства **Name** – это **идентификатор** объекта, его программное имя; а **Caption** – это надпись на объекте (заголовок).

Нанесите на форму три кнопки и запрограммируйте их на выполнение следующих действий:

- первая кнопка выводит в заголовок формы надпись «Самостоятельная работа по ЛАБОРАТОРНОЙ №3. Число.Месяц.Год».
- вторая кнопка вдвое увеличивает высоту и ширину формы, и одновременно заливает ее любимым (понравившимся Вам) цветом;
- третья кнопка восстанавливает первоначальный размер формы (т.е. уменьшает его в два раза), восстанавливает на форме серый цвет и в заголовок формы выводит ее имя (т.е. слово **MyForm**).

Контрольные вопросы к лабораторной работе №3.

1. Перечислите название *компонент*, находящихся на страничке *Палитры компонентов Standard* и их назначение.
2. Какое свойство позволяет *изменить месторасположение* компонента на форме. Перечислите все возможные *значения* этого свойства.
3. Какой компонент имеет свойство *WordWrap* и что это свойство обозначает.
4. Для чего нужна команда Save All.
5. Что обозначает свойство, **Width** и в каких единицах измерения оно представляет величину.
6. Что обозначает свойство, **Height**.
7. Чем отличаются свойства **Caption** и **Name**.
8. Как убрать/выставить *всплывающие подсказки* для определения названия компонент на *палитре компонентов*.

В тетради по практическим занятиям должны быть законспектированы краткие ответы на контрольные вопросы к лабораторной работе и записаны примеры всех программных кодов, использованных в данной лабораторной работе.

## Лабораторная работа №4

Тема: Работа с компонентами, расположенными на страничках *Палитры компонентов Standard* и *Additional*

Компоненты с вкладки **Additional** используются реже, но в некоторых случаях без них не обойтись. Среди них есть как компоненты для создания украшений, так и компоненты-модификации стандартных компонент.

**BitBtn** - *кнопка*, на которой помимо *текста* можно разместить *изображение*. За изображение отвечает свойство *Glyph* (формат - \*.bmp). Чтобы добавить картинку, следует щёлкнуть по кнопке с тремя точками в строке свойства *Glyph*  , в открывшемся окне нажать *Load...* и указать файл с картинкой.

**SpeedButton** - ещё один тип *кнопки*. Также поддерживает добавление изображения. Помимо этого кнопку можно сделать "плоской", т.е. без границ - за это отвечает свойство *Flat*.

**MaskEdit** - аналог *Edit*, но используется для ввода текста по маске. Например, можно указать ввод только цифр. Маска задаётся в свойстве *EditMask*. Имеется несколько стандартных вариантов.

**StringGrid** - *таблица*, в каждой ячейке которой может быть расположен текст. Часть ячеек можно сделать фиксированными (свойства *FixedCols* и *FixedRows*). **Количество строк** и **столбцов** таблицы задаётся соответственно свойствами *RowCount* и *ColCount*.

**DrawGrid** - таблица, но в ячейках помимо текста могут располагаться и другие объекты, например изображения.

**Image** - изображение, которое можно разместить на форме. Изображение задаётся в свойстве *Picture*. Свойство *Center* позволяет отцентрировать изображение, если размеров недостаточно для его полного отображения. Если установить свойство *AutoSize* в *true*, то размер компонента **Image** будет автоматически подгоняться под размер помещенной в него картинки. Если же свойство *AutoSize* установлено в *false*, то изображение может не поместиться в компонент или, наоборот, площадь компонента может оказаться много больше площади изображения. Свойство — *Stretch* позволяет подгонять не компонент под размер рисунка, а рисунок под размер компонента. Установите *AutoSize* в *false*, растяните или сожмите размер компонента **Image** и установите *Stretch* в *true*. Вы увидите, что рисунок займет всю площадь компонента, но поскольку вряд ли реально установить размеры **Image** точно пропорциональными размеру рисунка, то изображение исказится. Устанавливать *Stretch* в *true* может иметь смысл только для каких-то узоров, но не для картинок. Свойство *Stretch* не действует на изображения пиктограмм, которые не могут изменять своих размеров (*Stretch* позволяет задать сжатие/растяжение картинки под размеры компонента, *Proportional* указывает, следует ли при этом сохранять пропорции исходного изображения).

**Shape** - один из вариантов *геометрических фигур*. Тип фигуры определяется в свойстве *Shape*:

stCircle - окружность;

stEllipse - эллипс;

stRectangle - прямоугольник;

stRoundRect - прямоугольник с закруглёнными углами;

stRoundSquare - квадрат с закруглёнными углами;

stSquare - квадрат.

Свойства *Pen* и *Brush* позволяют задать стиль границ фигуры и её внутреннюю заливку.

**Bevel** - компонент для создания рельефа на форме - линий, окошек и т.д.

**ScrollBar** - прокручиваемая область (коробка с прокруткой), на которой можно разместить другие элементы.

**CheckBox** - аналог *ListBox*, но к каждой строке добавляется флажок. Таким образом более удобно выбирать значения из списка.

**Splitter** - *разделитель* для создания областей изменяемых размеров. Компонент следует разместить между двумя элементами, которые должны быть изменяемого размера. После этого следует настроить выравнивание (*Align*) элементов и *Splitter* автоматически начнёт работать.

**StaticText** - текстовая метка, аналог *Label*, но в отличие от него является полноценным элементом управления Windows. Использовать *StaticText* имеет смысл в том случае, если нужно разместить текст поверх какого-либо другого компонента. С *Label* этого сделать не удастся.

**ControlBar** - один из типов панелей инструментов. Автоматически перетаскивается по области, отведённой для панелей инструментов.

**ApplicationEvents** - используется для доступа к некоторым свойствам и событиям объекта *TApplication*. Этот объект - и есть само приложение. Работать с этим объектом можно и программно, но с помощью этого компонента всё же удобнее. Он является невидимым.

**ValueListEditor** - таблица из двух колонок - поля и значения. Некое подобие *StringGrid*.

**LabeledEdit** - *Label* и *Edit* «в одном флаконе». Оба компонента являются полностью настраиваемыми.

**ColorBox** - выпадающий список для выбора цвета.

**Chart** - мощный объект для построения диаграмм и графиков.

**ActionManager** - используется для управления действиями *TAction*. Аналогичен *ActionList* со страницы **Standard**, но имеет намного больше возможностей. При создании приложения на основе этой технологии рекомендуется использовать именно *ActionManager*.

**ActionMainMenuBar** - меню, работающее на основе *TAction*.

**ActionToolBar** - аналогично, панель инструментов на основе *TAction*.

**XPColorMap, StandardColorMap, TwilightColorMap** - стандартные цветовые схемы для объектов на основе *TAction* (кстати, они называются *Action-band* компонентами).

**CustomizeDlg** - диалог для настройки *Action-band* компонентов. Такой диалог используется, например в *Microsoft Word* для индивидуальной настройки панелей инструментов и меню. Помимо этого, *Action-band* компоненты позволяют сохранять и восстанавливать состояние всех объектов. Это делает приложение полностью настраиваемым.

## Выполнить.

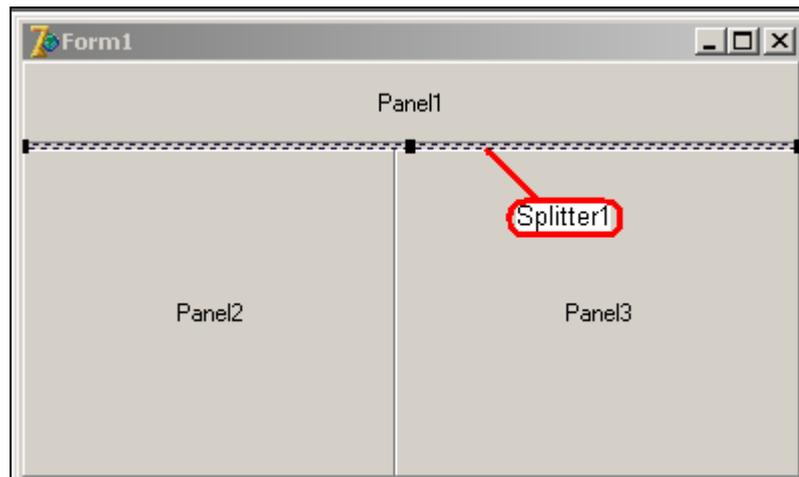
1. Нанесите на форму компонент Panel1 и установите для его свойства Align → alTop

**Внимание!!! Самостоятельно разберитесь со значениями свойства Align: alTop, alLeft, alClient, alBottom, alRight.** Это будет включено в контрольный опрос.

2. Нанесите на форму компонент Splitter1 и установите для его свойства Align → alTop

3. Нанесите на форму компонент Panel2 и установите для его свойства Align → alLeft

4. Нанесите на форму компонент Panel4 и установите для его свойства Align → alClient.



Запустите приложение и попробуйте с помощью компонента **Splitter** изменить размер панелей на форме.

Остановите приложение и сохраните проект.

5. На компонент Panel1 нанесите:

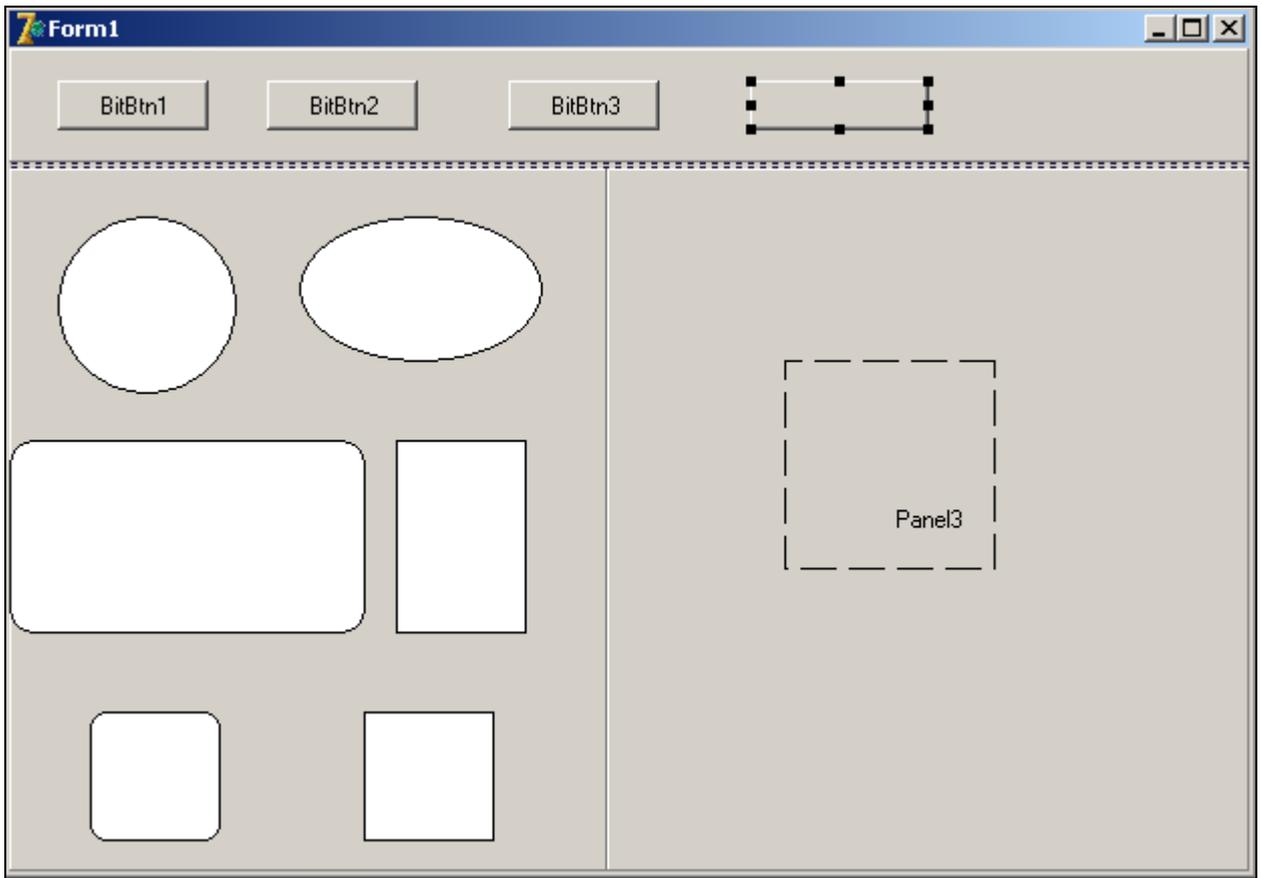
- три кнопки - BitBtn1, BitBtn2, BitBtn3;
- одну кнопку SpeedButton1;

На компонент Panel2 нанесите:

- шесть компонентов Share.

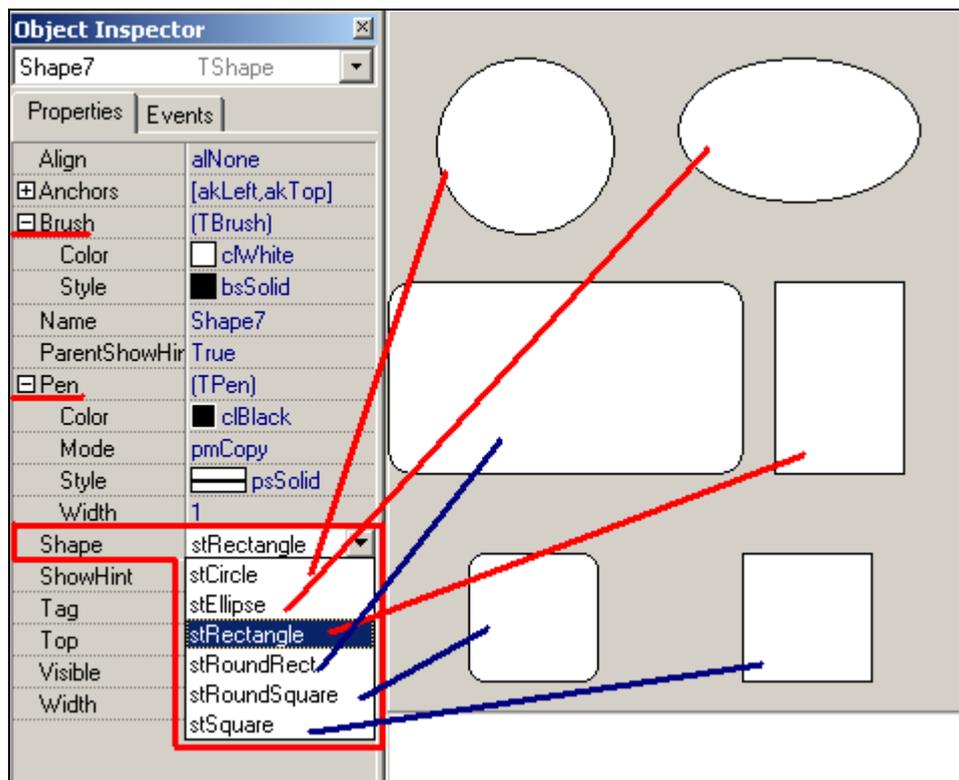
На компонент Panel3 нанесите:

- компонент Image1.



На рисунке ниже показано, какие значения *свойства Shape* у компонента **Shape** устанавливаются: круг, эллипс, прямоугольник, прямоугольник со скругленными углами, квадрат и квадрат со скругленными углами.

**Самостоятельно ознакомьтесь со сложными свойствами **Brush** и **Pen**.**



6. Запрограммируйте кнопку **BitBtn1** следующим кодом:

```
procedure TForm1.BitBtn1Click(Sender: TObject);  
begin  
Shape2.Brush.Color:=clRed;  
end;
```

7. Запустите приложение и проверьте его работу.

8. Остановите приложение и сохраните проект.

### 9. Работа с кнопкой **SpeedButton1**.

Установим залипание для кнопки **SpeedButton1**, для этого ее свойствам присвоим следующие значения: **AllowAllUp** → true; **GroupIndex** → 1 (или 2, или 3 – отличное от нуля число).

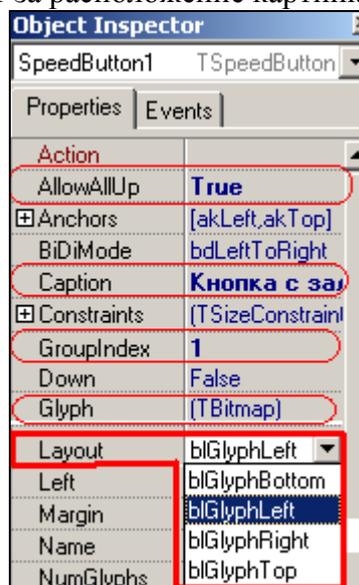
Запустите приложение и нажмите несколько раз на кнопку **SpeedButton1**, должен проявляться эффект залипания – для чего он нужен будет объяснено в дальнейших лабораторных работах..

Остановите приложение и сохраните проект.

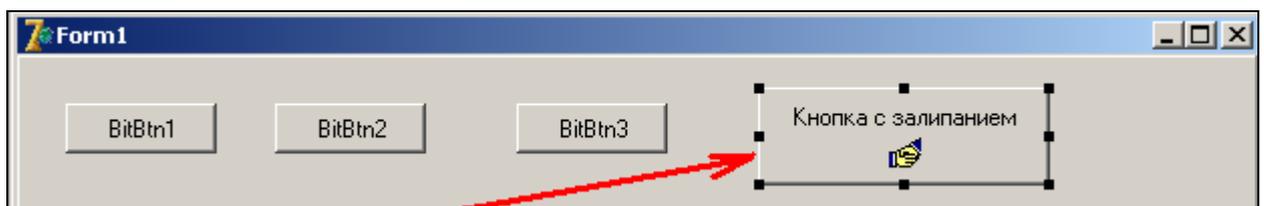
На все кнопки, у которых есть свойство **Glyph** можно подгрузить рисунок:

**Glyph** → кнопка **Load** → папка с рисунками → выбрать картинку → кнопка **Открыть** → кнопка **OK** .

Свойство **Layout** отвечает за расположение картинки на кнопке.



Оформите кнопку **SpeedButton1** в соответствии с нижеследующим рисунком:



**Не забывайте в процессе дизайна проекта периодически сохранять его!!!**

10. Подгрузите в компонент **Image1** рисунок:

**Picture** → кнопка **Load** → папка с рисунками → выбрать картинку → кнопка **Открыть** → кнопка **ОК** .

Для того, чтобы рисунок принял размер компонента **Image1** необходимо установить значения свойствам:

- **AutoSize** → *false*;
- **Stretch** → *true*.

Когда свойство **AutoSize** имеет значение *true*, то компонент **Image** принимает размер рисунка (свойство **Stretch** при этом не работает).

### Выполнить самостоятельно

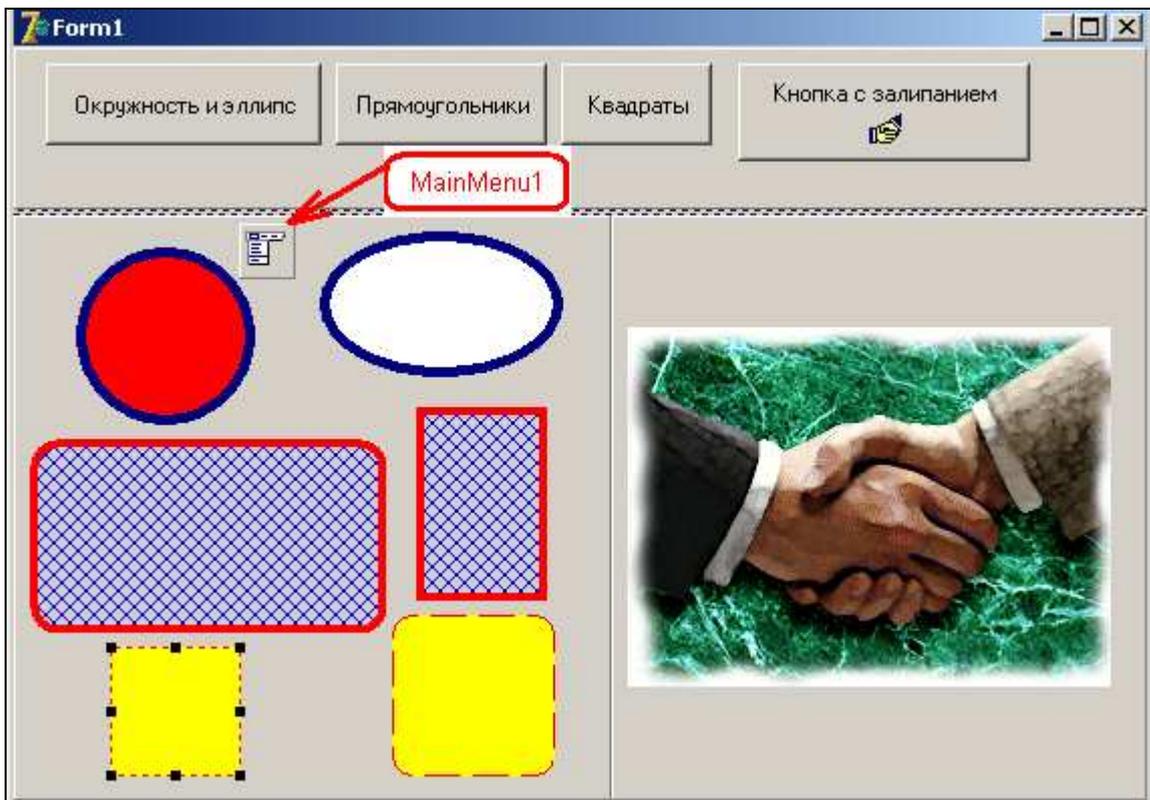
11. Для кнопки **BitBtn1** в уже существующую программу дописать команды, которые окрасят контур окружности и эллипса в синий цвет, а толщину контура установить равной 5. Саму кнопку подписать «Окружность и эллипс».

12. Для кнопки **BitBtn2** написать программу, которая заливает прямоугольники синим цветом, стиль заливки – диагональная штриховка, окрашивает контуры прямоугольников красным цветом, толщину контура устанавливает равной 4. Саму кнопку подписать «Прямоугольники».

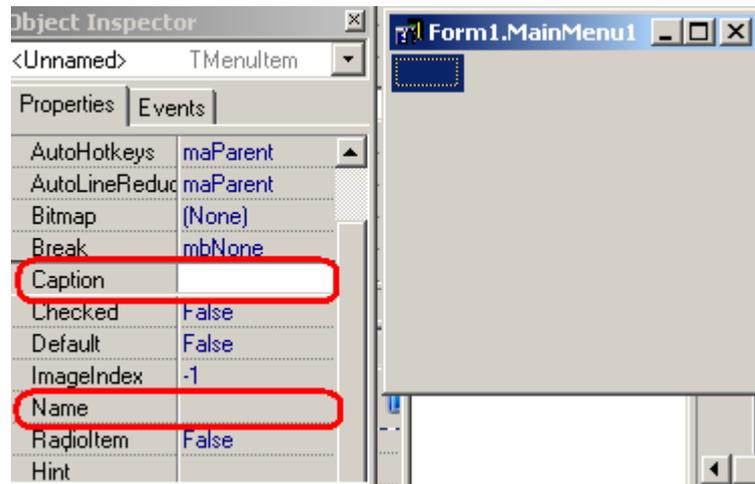
13. Для кнопки **BitBtn3** написать программу, которая заливает квадраты желтым цветом, окрашивает их контуры красным цветом, а стиль линию контура делает прерывистой. Саму кнопку подписать «Прямоугольники».

### Создание главного меню приложения

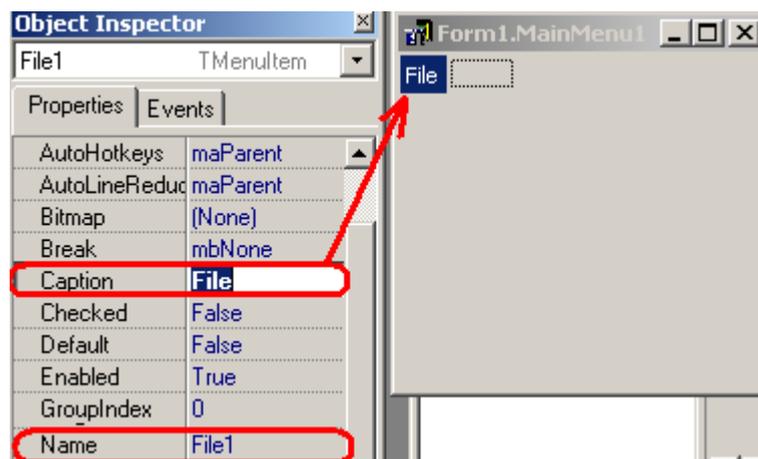
14. Нанесите на форму компонент **MainMenu1**.



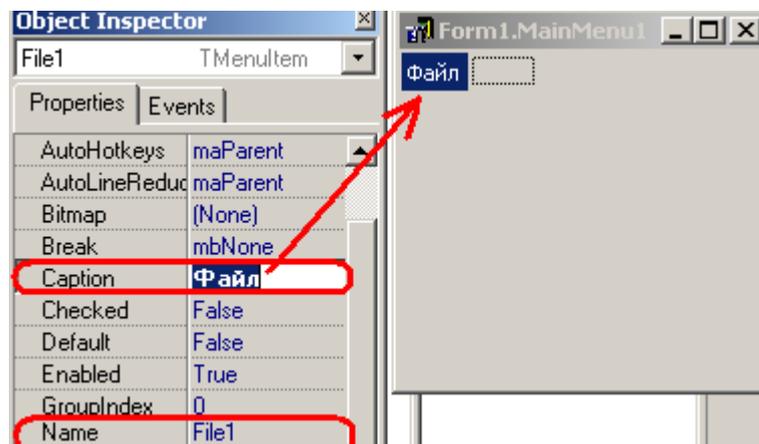
15. Выполните по компоненту двойной щелчок. Это действие вызовет окно *Редактора меню* и позволит Вам сформировать *главное меню* приложения.



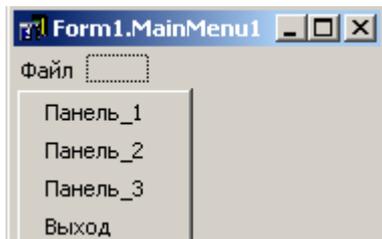
16. Шаг 1. Введите значение для свойства **Caption** → **File**. Автоматически для свойства Name присвоится это же значение с добавленным *индексом 1* → **File1**.



17. Шаг 2. Вернитесь к свойству **Caption** и русифицируйте пункт меню **Caption** → **Файл**.



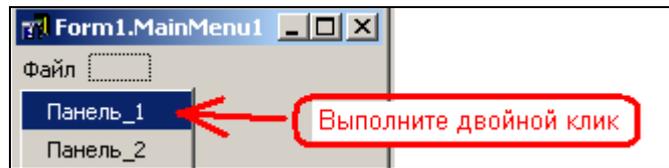
18. Во всех Windows-приложениях пункты *главного меню* раскрываются и имеют соответствующие *подпункты*. Мы для нашего меню *Файл* создадим четыре подпункта:



- **Панель\_1.** Нажатие этого пункта закрасит компонент Panel1 в синий цвет.



Для «оживления» пункта меню его необходимо запрограммировать. Выполните двойной щелчок по созданному пункту **Панель\_1**.



В открывшуюся заготовку (она называется *обработчик событий*) внесите программу:

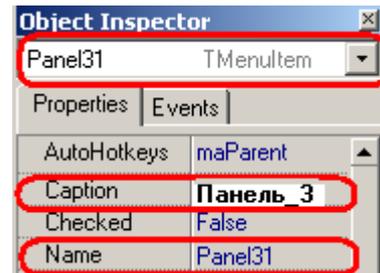
```
procedure TForm1.Panel1Click(Sender: TObject);
begin
  Panel1.Color:=clBlue;
end;
```

- **Панель\_2.** Нажатие этого пункта закрасит компонент Panel2 в зеленый цвет.



Этот пункт меню оживите самостоятельно.

- **Панель\_3.** Нажатие этого пункта закрасит компонент Panel3 в желтый цвет.



Этот пункт меню оживите самостоятельно.

- **Выход.** Нажатие этого пункта *остановит* наше приложение.



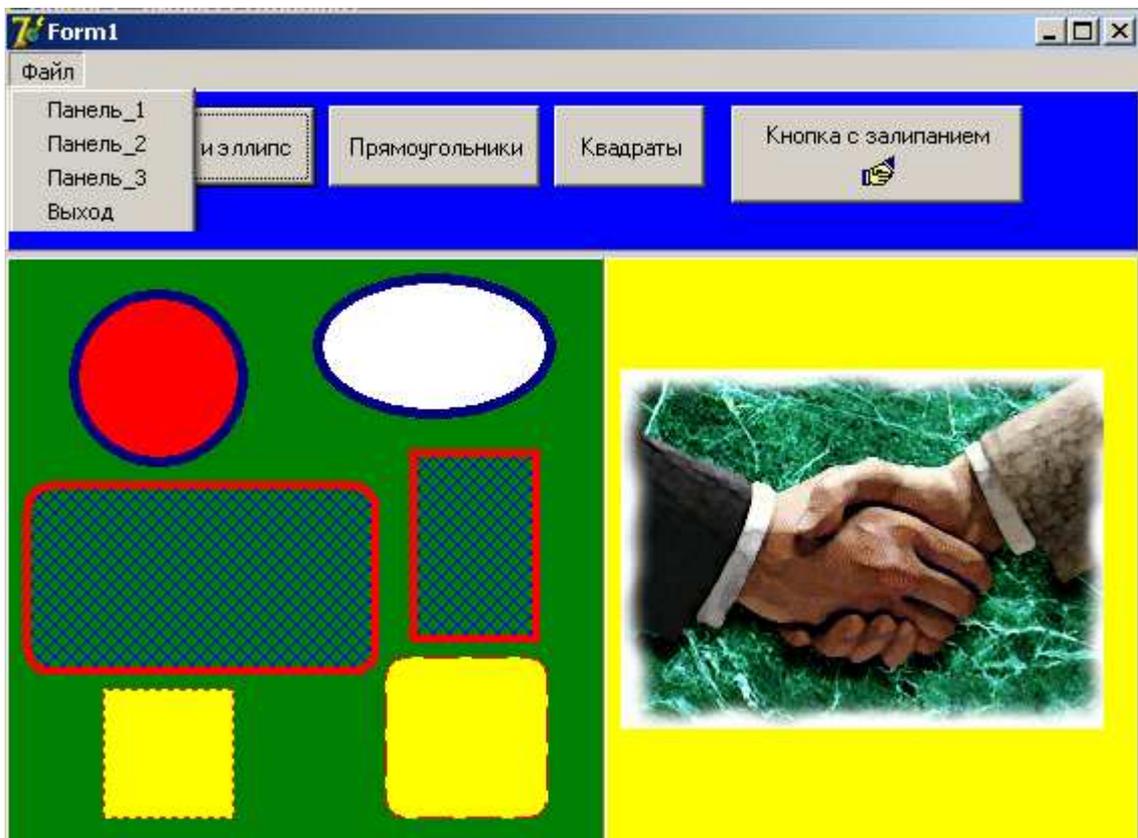
Программная строка для пункта «Выход» содержит всего одну команду:

```

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

```

Запустите созданное приложение и проверти работу подпунктов *главного меню*.



## Контрольные вопросы к лабораторной работе №4.

1. Назначение компонентов: **BitBtn**, **SpeedButton**, **Image**, **Splitter**, **Shape** и их основные свойства.
2. Какой эффект создает кнопка **SpeedButton1** и какие свойства его формируют.
3. Какое свойство отвечает за *месторасположение* рисунка на кнопке.
4. Для чего служит свойство **Stretch** у компонента **Image**.
5. Как подгрузить картинку на кнопку или в компонент **Image**.
6. Объясните значения свойств Align: alTop, allBottom и т. д.
7. Расскажите о свойствах **Brush** и **Pen** компонента **Shape**.
8. Расскажите о формировании *главного меню* приложения и его подпунктов.

Все программные коды данной лабораторной работы законспектировать в тетрадь с пояснениями выполняемых ими действий.

## Лабораторная работа №5

### Тема. Основные свойства компонентов. Создание проекта с несколькими формами.

Свойства являются *атрибутами* компонента, определяющими его внешний вид и поведение. Многие свойства компонента в колонке свойств имеют значение, устанавливаемое по *умолчанию* (например, высота кнопок). Свойства компонента отображаются на странице свойств (Properties) в *инспекторе объектов*. Инспектор объектов отображает *опубликованные* (published) свойства компонентов. Помимо published-свойств, компоненты могут и чаще всего имеют *общие* (public), опубликованные свойства, которые доступны только во время выполнения приложения. *Инспектор объектов* используется для установки свойств во время проектирования. Можно определить свойства во время проектирования или написать код для видоизменения свойств компонента во время выполнения приложения.

Компонентами в Delphi называются потомки класса TComponent. Все компоненты Delphi порождены от класса TComponent, в котором инкапсулированы самые общие свойства и методы компонентов. Предком TComponent является класс TPersistent, который произошел непосредственно от базового класса TObject.

#### Наиболее общие и часто встречающиеся свойства компонентов:

**Name** - идентификатор объекта, имя компонента, по которому к нему можно обращаться из программы..

**Caption**- надпись на объекте

**Text** – строка, связанная с компонентом. Свойство предназначено для работы со списком как с единой строкой текста, представляющее все содержимое списка в качестве одной строки. При этом в тех местах, где заканчиваются реальные строки списка, добавляются символы новой строки и возврата каретки (#10 и #13). Применительно к компоненту

**Memo**, его собственное свойство **Text** содержит точно те же данные, что и **Lines.Text**.

**Font** – шрифт текста, сложное свойство с множеством вложенных в него свойств.

**Color** - цвет в формате \$RRGGBB

**Visible** – видимость объекта, отображать ли компонент.

**Alignment** - определяет выравнивание текста по горизонтали в области метки. Может принимать значения taLeftJustify, taRightJustify и taCenter

**Layout** - определяет выравнивание текста по вертикали. Может принимать значения tlTop, tlCenter и tlBottom

**Align** - способ выравнивания компонента

alNone - не использовать

alTop- по верхнему краю, шириной на весь контейнер

alBottom - по нижнему краю, шириной на весь контейнер

alLeft - по левому краю, высотой на весь контейнер

alRight - по правому краю, высотой на весь контейнер

alClient - во весь контейнер

alCustom - размеры и положения определяются разработчиком.

**Enabled** – доступность объекта, активность (способность реагировать на действия пользователя).

**Height** - высота компонента

**Width** - ширина компонента

**Top** – координата сверху

**Left** - координата слева.

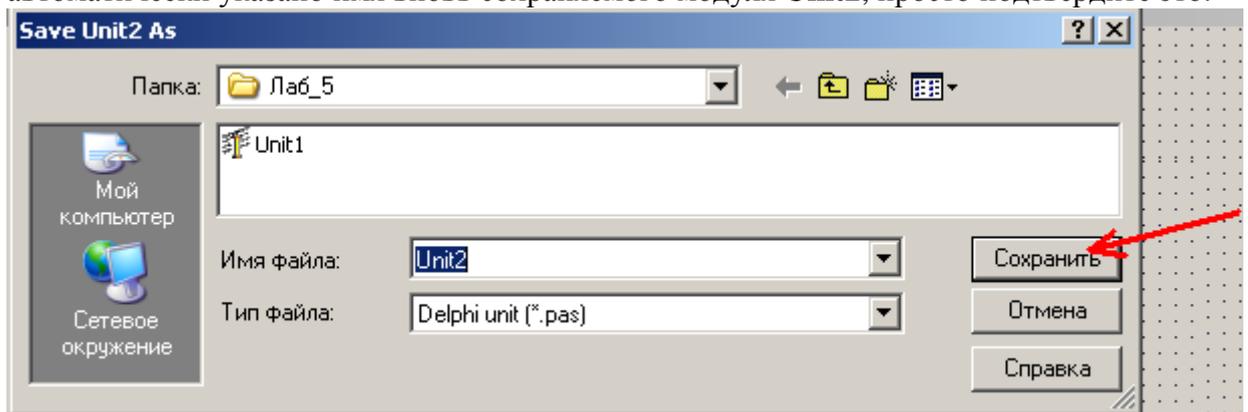
**AutoSize** - определяет, должен ли компонент изменять свои размеры в зависимости от содержимого (например, текста)

**Transparent** - определяет, должен ли фон метки быть прозрачным

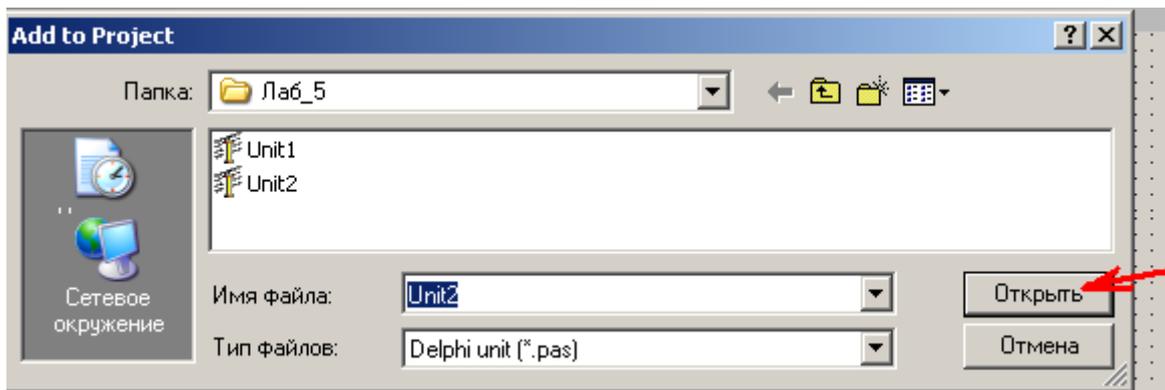
**WordWrap** - определяет, должен ли текст, не помещающийся по ширине, переноситься на следующие строки.

### Создание проекта с двумя формами

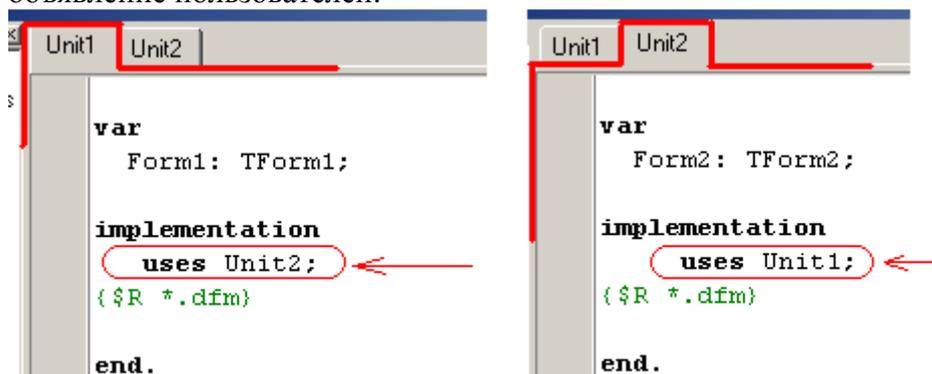
1. Создайте папку «Лабораторная работа №5», откройте среду Delphi и сохраните в созданную папку еще пустой проект.
2. Далее выполните следующие действия
  - вызовите новую (вторую) форму: File → New → Form; на экране появится новая форма - **Form2**.
  - сохраните вторую форму File → Save. В открывшемся *окне сохранения* будет автоматически указано имя вновь сохраняемого модуля **Unit2**, просто подтвердите это:



3. Подсоедините новую форму к проекту следующим образом: меню Project → Add to Project и нажмите кнопку **Открыть** .



4. Далее в окне *редактора кода* в секции **implementation** каждого из модулей сделайте объявление пользователей:



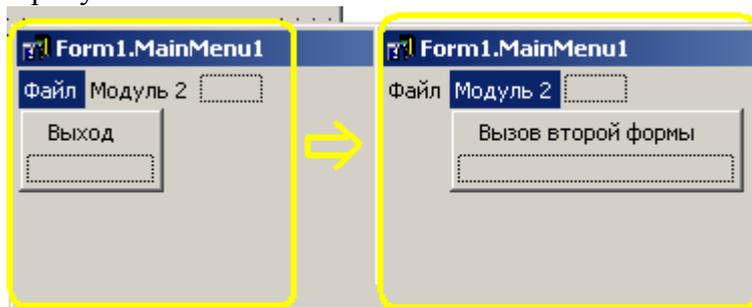
В первом модуле Unit1 объявите нового пользователя - модуль 2, т.е. Unit2

А во втором модуле - Unit2 объявите пользователя Unit1

Если новые модули **Unit**-ы не объявлять в секции **implementation**, то они не будут «видеть» друг друга и работа одной *формы* с другой будет **невозможна**.

5. Запустите приложение (на экране будет только пустая первая форма Form1). Остановите его и сохраните проект, если в нем нет ошибок.

6. Нанесите на первую форму - **Form1** компонент **MainMenu1** и создайте в главном меню пункты в соответствии с рисунком:

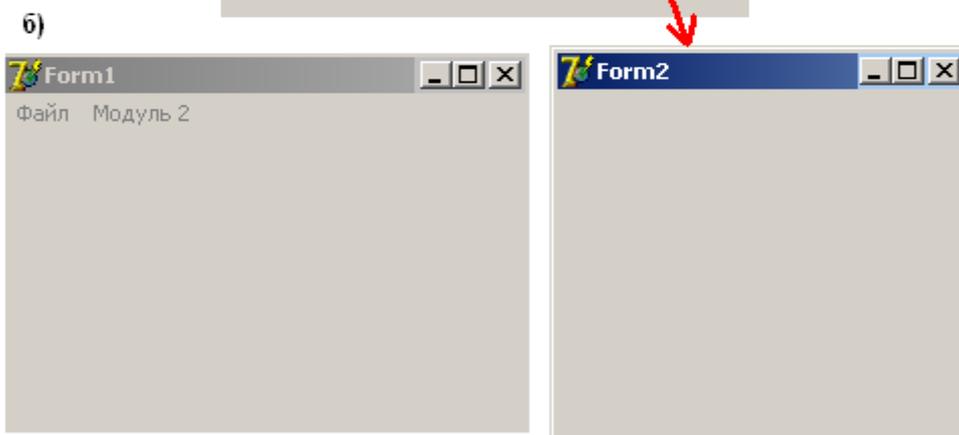
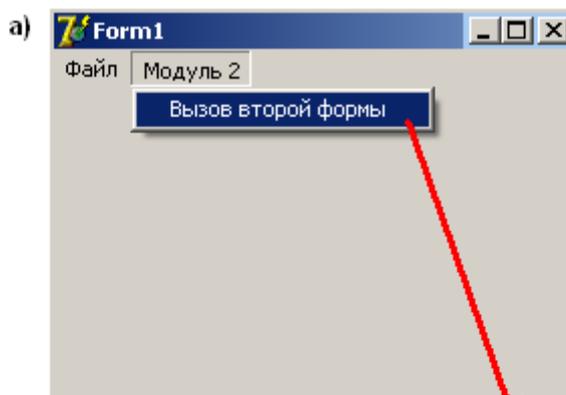


При нажатии на подпункт «Выход» приложение должно закрываться, а при нажатии на подпункт «Вызов второй формы» на экране должна появляться вторая форма - **Form2**.

Пункт «Выход» запрограммируйте самостоятельно, а в обработчик события `onClick` подпункта «Вызов второй формы» внесите следующую программу:

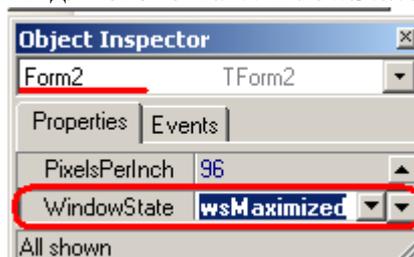
```
Unit1 | Unit2 |
procedure TForm1.Forma21Click(Sender: TObject);
begin
  Form2.Show;
end;
```

7. Запустите приложение проверьте работу всех пунктов главного меню.



8. Остановите приложение и досохраниите проект.

9. Установите для второй формы для свойства **WindowState** значение **wsMaximized**



Поставьте на вторую форму две кнопки и запрограммируйте их соответственно:  
( Напоминание, добраться до второй формы можно через *менеджер проекта*)



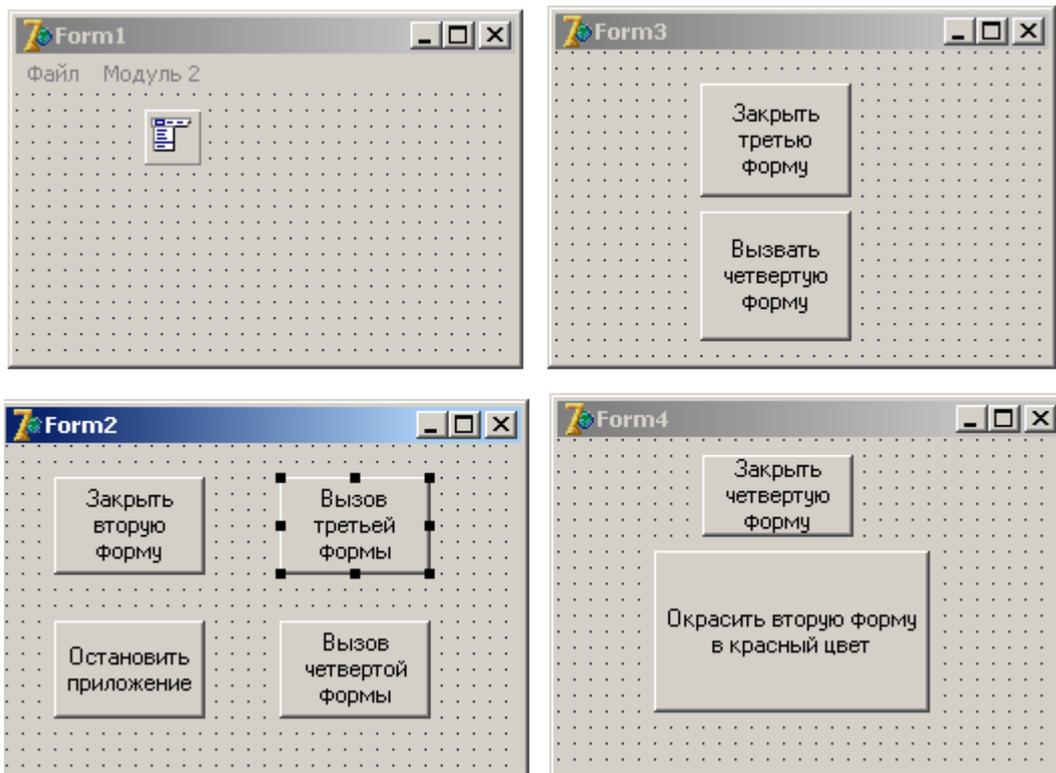
```
Unit1 Unit2
procedure TForm2.Button1Click(Sender: TObject);
begin
  Form2.Close;
end;

procedure TForm2.Button2Click(Sender: TObject);
begin
  Application.Terminate;
end;
```

10. Самостоятельно добавьте к проекту ещё две формы и доработайте проект в соответствии с нижеследующими рисунками и в соответствии с надписями на кнопках самостоятельно запрограммируйте кнопки:

**Внимание!!!** Не забывайте объявлять новых пользователей в секции **implementation** каждого из модулей.

```
Unit1 Unit2 Unit3 Unit4
implementation
  uses Unit2, Unit1, Unit4;
{$R *.dfm}
```



Контрольные вопросы.

1. Перечислите *свойства* компонентов, которые чаще всего используются при дизайне проекта и кратко расскажите о функциональном назначении этих свойств.
2. Для чего объявляются новые пользователи в секции **implementation** каждого из модулей проекта.
3. Перечислите этапы подключения к проекту новой формы.

Лабораторная работа № 6 - 7  
**Тема: Компоненты графики Image и PaintBox**

Компоненты **Image**, **PaintBox**, а также и сама форма **Form** представляют собой некоторую ограниченную поверхность с *канвой*, на которую можно заносить изображения. Преимущество рисования на **PaintBox** и **Image** вместо формы **Form** лишь в некотором облегчении расположения одного или нескольких рисунков в площади окна, т.е. отпадает необходимость сложных и нудных расчетов координат канвы формы, обеспечивающих требуемое взаимное расположение изображений.

**Канва** (*Canvas*, холст) не является *компонентом*. **Канва** представляет собой *область компонента*, на которой можно рисовать или отображать готовые изображения. Она содержит свойства и методы, существенно упрощающие графику Delphi. Все сложные взаимодействия с системой спрятаны для пользователя.

Каждая точка **канвы** имеет координаты **X** и **Y**. Система координат канвы, как и везде в Delphi, имеет началом **левый верхний угол** канвы. Координата **X** возрастает при перемещении слева направо, а координата **Y** — при перемещении сверху вниз.

Координаты измеряются в *пикселях*. **Пиксель** — это наименьший элемент поверхности рисунка, с которым можно манипулировать. Важнейшее свойство пикселя — его цвет.

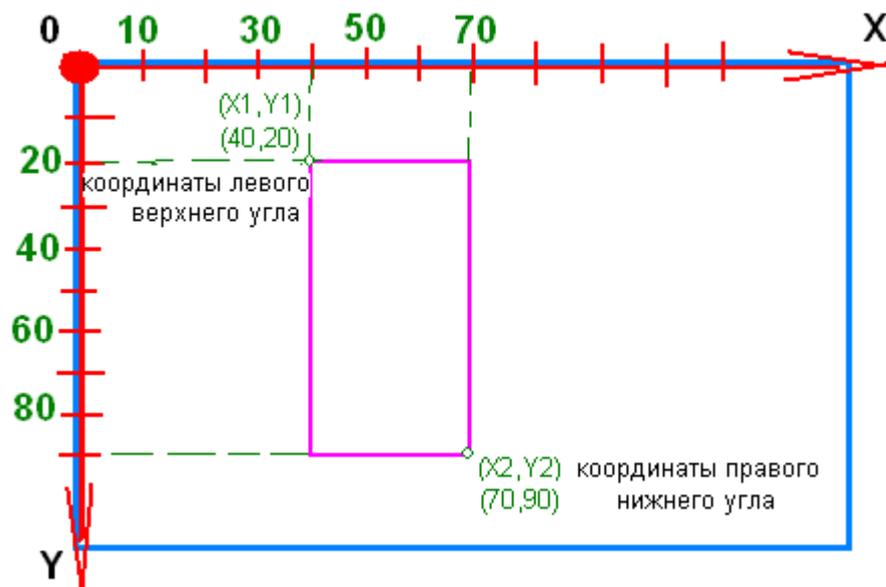


Рис. 8

Канва имеет свойство **Pixels**. Это свойство представляет собой двумерный массив, который отвечает за цвета канвы. Например, `Canvas.Pixels[10,20]` соответствует цвету пикселя, 10-го слева и 20-го сверху. С массивом пикселей можно обращаться как с любым свойством: изменять цвет, задавая пикселю новое значение, или определять его цвет по хранящемуся в нем значению. Например, `Canvas.Pixels[10,20] := 0` или `Canvas.Pixels[10,20] := clBlack` — это задание пикселю черного цвета.

Канва — объект класса `TCanvas` имеет множество методов, которые позволяют рисовать графики, линии, фигуры с помощью свойства **Pen** — перо. Это свойство является объектом, в свою очередь имеющим ряд свойств. Одно из них уже известно вам свойство **Color** — цвет, которым наносится рисунок (граница рисунка). Второе свойство — **Width** (ширина линии). Ширина задается в *пикселях*. По умолчанию ширина равна 1.

Свойство **Style** определяет вид линии. Это свойство может принимать следующие значения:

**psSolid** - сплошная линия

**psDash** - штриховая линия

**psDot** - пунктирная линия

**psDashDot** - штрих-пунктирная линия

**psDashDotDot** - линия, чередующая штрих и два пунктира

**psClear** - отсутствие линии

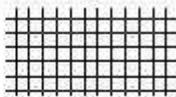
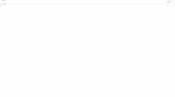
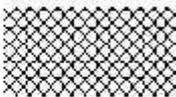
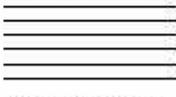
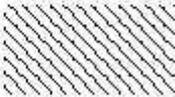
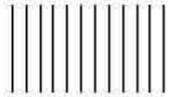
**psInsideFrame** - сплошная линия, но при **Width** > 1 допускающая цвета, отличные от палитры Windows.

### Свойство Brush

Фигуры в общем случае рисуются не пустыми, а *закрашенными* с помощью свойства канвы **Brush** — кисть. Атрибуты объекта `Brush` можно изменять, используя свойства `Color` и `Style`. Свойство **Color** определяет цвет фона компонента.

| Константа    | Значение цвета                  |
|--------------|---------------------------------|
| clBlack      | Черный                          |
| clMaroon     | Темно-бордовый                  |
| clGreen      | Зеленый                         |
| clOlive      | Оливково-зеленый                |
| clNavy       | Темно-синий                     |
| clPurple     | Пурпурный                       |
| clTeal       | Морской воды                    |
| clGray       | Серый                           |
| clSilver     | Серебряный                      |
| clRed        | Красный                         |
| clLime       | Лимонно-зеленый                 |
| clBlue       | Синий                           |
| clYellow     | Желтый                          |
| clFuchsia    | Сиреневый                       |
| clAqua       | Голубой                         |
| clWhite      | Белый                           |
| clBackground | Текущий цвет фона стола Windows |
| clScrollBar  | Текущий цвет полос прокрутки    |

Свойство **Style** определяет шаблон заполнения (штриховку). По умолчанию значение **Style** равно **bsSolid**, что означает сплошное закрашивание цветом **Color**.

| Значение    | Шаблон  | Значение     | Шаблон  |
|-------------|---|--------------|---|
| bsSolid     |  | bsCross      |  |
| bsClear     |  | bsDiagCross  |  |
| bsBDiagonal |  | bsHorizontal |  |
| bsFDiagonal |  | bsVertical   |  |

У канвы имеется *свойство PenPos* типа TPoint (см.). Это свойство определяет в координатах канвы текущую позицию пера. Перемещение пера без прорисовки линии, т.е. изменение PenPos, производится методом канвы **MoveTo(X,Y)**. Здесь (X, Y) — координаты точки, в которую перемещается перо. Эта текущая точка становится исходной, от которой методом LineTo(X,Y) можно провести линию в точку с координатами (X,Y). При этом, текущая точка перемещается в конечную точку линии и новый вызов LineTo будет проводить точку из этой новой текущей точки.

Перо может рисовать не только прямые линии, но и фигуры. Полный список методов канвы, использующих перо приведен в таблице.

Таблица Примеры методов для рисования на канве

| Наименование     | Рисование   |
|------------------|---|
| <b>Arc</b>       | <p>Рисует дугу <i>окружности</i> или <i>эллипса</i><br/> Arc(x1,y1,x2,y2,x3,y3,x4,y4: Integer)<br/> Метод Arc рисует дугу окружности или эллипса с помощью текущих параметров пера Pen. Точки (x1,y1) и (x2,y2) определяют прямоугольник, описывающий эллипс. Начальная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку (x3,y3). Конечная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку (x4,y4). Дуга рисуется против часовой стрелки от начальной до конечной точки.</p>   |
| <b>Chord</b>     | <p>Рисует заполненную замкнутую фигуру, ограниченную дугой окружности или эллипса и хордой<br/> Chord (x1,y1,x2,y2,x3,y3,x4,y4:Integer);<br/> Метод <b>Chord</b> рисует замкнутую фигуру: <i>дугу</i> окружности или эллипса, <i>замкнутую хордой</i>, с помощью текущих параметров пера Pen. Фигура заполняется текущим значением Brush. Точки (x1,y1) и (x2,y2) определяют прямоугольник, описывающий эллипс. Начальная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку (x3,y3). Конечная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку (x4,y4). Дуга рисуется против часовой стрелки от начальной до конечной точки. Хорда соединяет точки(x3,y3) и (x4,y4).</p> |
| <b>Ellipse</b>   | <p>Рисует заполненную окружность или эллипс<br/> Ellipse (x1,y1,x2,y2:Integer);</p>   |
| <b>FillRect</b>  | <p>вычерчивает <i>закрашенный</i> прямоугольник, используя в качестве инструмента только кисть (Brush):<br/> Form1.Canvas.Brush.Color := clRed;<br/> Form1.Canvas.FillRect(Rect(X1,Y1,X2,Y2)) →<br/> Form1.Canvas.FillRect(Rect(10,10,40,60))</p>   |
| <b>FrameRect</b> | <p>вычерчивает только <i>границы</i> прямоугольника, используя в качестве инструмента только кисть (Brush):<br/> Form1.Canvas.Brush.Color := clGreen;<br/> Form1.Canvas.FrameRect(Rect(X1,Y1,X2,Y2))</p>  |
| <b>LineTo</b>    | <p>Рисует на канве прямую линию, начинающуюся с текущей позиции пера и кончающуюся указанной точкой.<br/> LineTo (x,y:Integer);</p>   |
| <b>Polygon</b>   | <p>Метод Polygon рисует на канве замкнутую фигуру (полигон, многоугольник) по множеству угловых точек, заданному массивом Points. Первая из указанных точек соединяется прямой с последней. Этим метод Polygon отличается от метода Polyline, который не замыкает конечные точки. Рисование проводится текущим пером Pen. Внутренняя область фигуры закрашивается текущей кистью Brush.</p>   |
| <b>PolyLine</b>  | <p>Рисует на канве текущим пером кусочно-линейную кривую по заданному множеству точек.</p>  |

|                  |   |
|------------------|---|
| <b>Rectangle</b> | Рисует на канве текущим пером прямоугольник и закрашивает его текущей кистью.   |
| <b>RoundRect</b> | Метод RoundRect рисует на канве прямоугольную рамку со скругленными углами, используя текущие установки пера Pen и заполняя площадь фигуры текущей кистью Brush. Рамка определяется прямоугольником с координатами углов (x1,y1) и (x2,y2). Углы скругляются с помощью эллипсов с шириной x3 и высотой y3, RoundRect(x1,y1,x2,y2,x3,y3:Integer);<br>RoundRect( 20, 20, 50, 50, 3, 3); |

Компонент **PaintBox** (вкладка System) класса **TPaintBox** применяется в тех случаях, когда необходимо иметь прямоугольную область для выполнения графических операций, используя его свойство *Canvas*. При этом можно использовать все возможности канвы – карандаш, кисть, шрифт и методы построения геометрических фигур. Использование этого компонента может быть альтернативой рисованию по канве формы **Form**, но **TPaintBox** не позволяет загружать готовые изображения.

Само рисование программируется в обработчике события **onPaint** и тогда, при запуске приложения, в графическом компоненте сразу прорисовывается запрограммированная фигура.

### ВЫПОЛНИТЬ:

Создайте папку «Лабораторная\_б» и в нее сохраните пустой проект. Все пункты по рисованию геометрических фигур с **1** по **6** должны быть отработаны в Вашем проекте.

#### 1. Рисование линии.

Для рисования *линии* необходимо проделать два шага:

- поставить *перо* в начальную точку (в то место на экране, с которого будет начинаться линия).
- провести линию до конечной точки

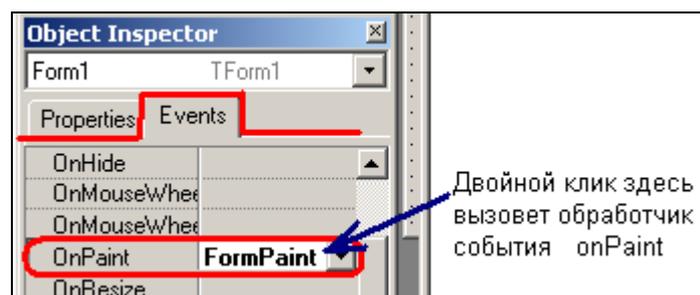
После рисования линии перо остается в конечной точке.

Рассмотрим методы для осуществления этих действий:

**Canvas.MoveTo(x,y)**- данный метод устанавливает перо в точку с координатами (x, y), не рисуя при этом никаких линий.

**Canvas.LineTo(x,y)** – данный метод рисует линию из точки, в которой находится перо, в новую точку с координатами (x, y).

В обработчик события формы **OnPaint** (прорисовка) вставьте следующий код:



```

procedure TForm1.FormPaint(Sender: TObject);
begin
  Form1.Canvas.MoveTo(10, 20); // ставим перо в начало линии
  Form1.Canvas.LineTo(50, 80); // рисуем линию
end;

```

Запустите приложение. Остановите. Досохраните проект.

По умолчанию, *точка начала* рисования установлена в (0,0), то есть, если сразу вызвать метод **Объект.Canvas.LineTo(100,100)**; то будет нарисована линия из точки (0,0) в точку (100, 100). Точка начала рисования автоматически переместится в (100, 100), то есть, если выполнить команду " **Объект.Canvas.LineTo(200, 100);**", то следующая линия будет нарисована из точки (100, 100) в (200, 100). Поэтому, если мы хотим рисовать линии несоединённые друг с другом, то придётся воспользоваться методом **MoveTo**.

**MoveTo** перемещает точку начала рисования линии в указанные координаты x и y

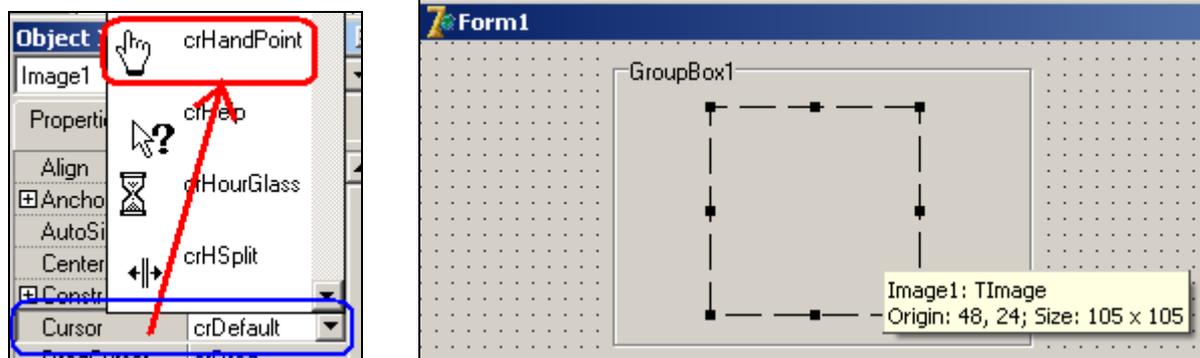
## 2. Рисование прямоугольника

*Прямоугольник* можно нарисовать и при помощи линий (метод **LineTo(x,y)**), но для его рисования существует специальный метод **Canvas.Rectangle(x1, y1, x2, y2)** - рисует прямоугольник по двум диагональным точкам (рис.б.1). Координаты первой точки (x1, y1), координаты второй (x2, y2). Порядок указания точек не имеет значения.

Для того чтобы создаваемое приложение было наглядным, нанесите на форму компонент **GroupBox1** а в него поместите компонент **Image1**.

Для компонента **GroupBox1** измените значение свойства **Caption** → ПРЯМОУГОЛЬНИК.

Для компонента **Image1** измените внешний вид курсора (указателя мышки) при его наведении на компонент (свойство **Cursor** → **crHandPoint**):



У компонента **Image** нет обработчика события **OnPaint**, поэтому внести код для прорисовки прямоугольника можно в один из имеющихся у компонента **Image** обработчиков событий, например, в **OnMouseDown**.

```

procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Image1.Canvas.Rectangle(100, 5, 60, 70);
end;

```

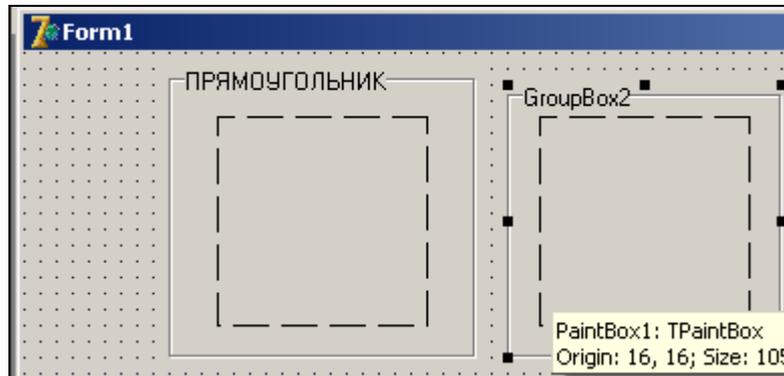
Запустите приложение. Нажмите клавишу мышки при подведении ее к компоненту **Image**. Остановите приложение. Досохраните проект.

### 3. Рисование эллипса (овала)

Нанесите на форму компонент **GroupBox2**, а в него поместите компонент **PaintBox1**.

Для компонента **GroupBox2** измените значение свойства **Caption** → ЭЛЛИПС.

Для компонента **PaintBox1** измените внешний вид курсора (указателя мышки) при его наведении на компонент (свойство **Cursor** → **crHandPoint**):



У компонента **PaintBox** обработчик события **OnPaint** есть, но для приобретения навыка внесем программу по прорисовке эллипса в обработчик событий в **On MouseUp** (момент отпущения клавиши мышки – *клавиша вверх*).

Рисование эллипса происходит с помощью специального метода объекта Canvas под названием **Ellipse**.

**Canvas.Ellipse(x1, y1, x2, y2)** – рисует эллипс по двум диагональным точкам прямоугольника в, который будет вписан эллипс. Естественно сам прямоугольник не отображается.

```
procedure TForm1.PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;  
  Shift: TShiftState; X, Y: Integer);  
begin  
  PaintBox1.Canvas.Ellipse(100, 5, 60, 70);  
end;
```

Чтобы оформить цветовую гамму нарисованной фигуры необходимо применять основные свойства пера и заливку:

```
Объект.Canvas.Pen.Width := 4  
Объект.Canvas.Pen.Color := clLime  
Объект.Canvas.Brush.Color := clRed
```

Если при рисовании эллипса координаты **X** и **Y** у углов будут совпадать, то получится окружность - `Объект.Canvas.Ellipse(0,0,50,50);`

### 4. Рисование полигона (заполненного многоугольника)

Для прорисовки полигона ниже представлено два способа.

Первый способ предполагает *объявление массива точек* (массив с именем **P**) и задание координат этим точкам - рисует на канве текущим пером замкнутую фигуру

(многоугольник) по заданному множеству угловых точек, замыкая первую и последнюю точки и закрашивая внутреннюю область фигуру текущей кистью.

```
procedure TForm1.PaintBox2MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  P: array[1..4] of TPoint; //массив точек
begin
  //координаты точек
  P[1].X:=10; P[1].Y:=10;
  P[2].X:=50; P[2].Y:=10;
  P[3].X:=80; P[3].Y:=40;
  P[4].X:=60; P[4].Y:=90;
  //заливка цветом
  PaintBox2.Canvas.Brush.Color:=clGreen;
  //рисует полигон
  PaintBox2.Canvas.Polygon(P);
end;
```

Второй способ рисует тот же полигон, но массив узловых точек задан другим образом:

```
procedure TForm1.PaintBox2MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  PaintBox2.Canvas.Brush.Color:=clGreen;
  PaintBox2.Canvas.Polygon([Point(10, 10), Point(50, 10), Point(80, 40),
    Point(60, 90)]);
end;
```

Нанесите на форму еще один компонент **PaintBox2** и запрограммируйте его на прорисовку полигона любым, из представленных двух, способом.

Прорисовка *незаполненного многоугольника (PolyLine)* подобна построению полигона.

## 5. Прорисовка текста

Ещё есть очень нужная функция **TextOut**, которая позволяет рисовать текст, используя шрифт, заданный в канве: **TextOut** - рисует данную строку на Canvas начиная с координат (x,y) - фон текста заполняется текущим цветом кисти:

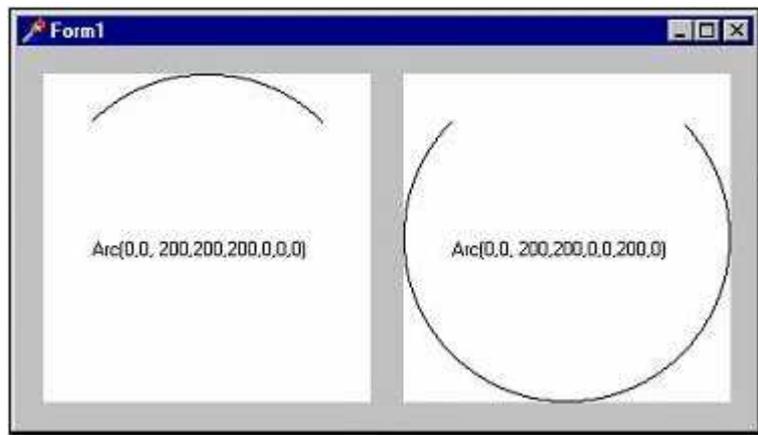
```
Объект.Canvas.TextOut(10, 10, 'УЧИМСЯ РИСОВАТЬ');
```

Самостоятельно добавьте текст к линии, которая рисуется на форме.

## 6. Прорисовка дуги и хорды

### Дуга

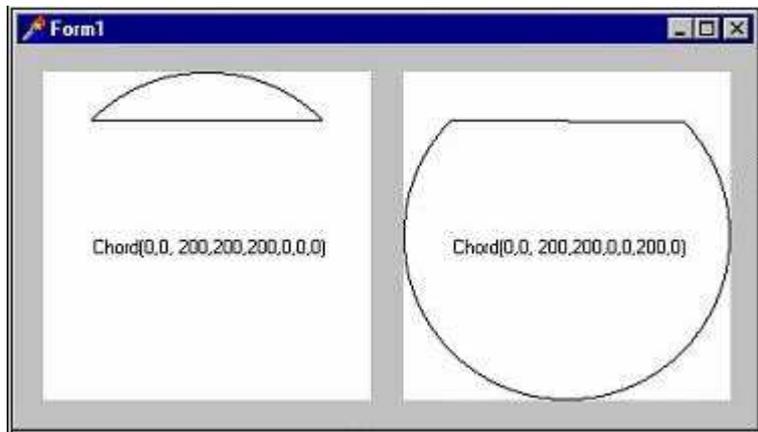
```
Image1.Canvas.Arc(0,0, 200,200, 200,0, 0,0);
Image2.Canvas.Arc(0,0, 200,200, 0,0, 200,0);
```



**Хорда.**

Image1.Canvas.Chord(0,0, 200,200, 200,0, 0,0);

Image2.Canvas.Chord(0,0, 200,200, 0,0, 200,0);



В качестве самостоятельной работы по лабораторным работам 6-7 выполнить приложение в соответствии с нижеследующим рисунком:

Form1

GroupBox1



Рисунок 1

GroupBox2

КВАДРАТ И ЭЛЛИПС

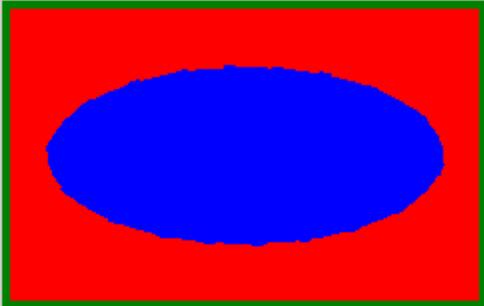


Рисунок 2

GroupBox3

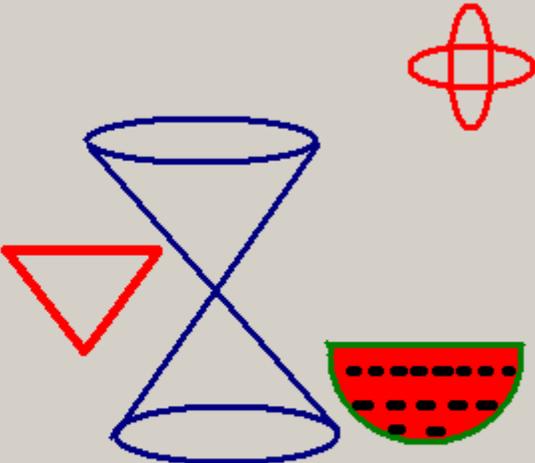


Рисунок 3

Основные теоретические моменты лабораторной работы и фрагменты программ по построению фигур должны быть законспектированы в тетради по практическим занятиям.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if edit1.Text='квадрат' then
  begin
    InvalidateRect(0, nil, true); //очистка канвы
    paintbox1.Canvas.Rectangle(100,5,60,70);
  end;
  if edit1.Text='круг' then
  begin
    InvalidateRect(0, nil, true); //очистка канвы
    paintbox1.Canvas.Ellipse(100,5,50,50);
  end;
end;
```

Способы очистки канвы

//это очистит рабочий стол

```
InvalidateRect(0, nil, true);
```

//

```
InValidateRect(Canvas.Handle,nil,true);
```

// Если вы используете холст формы, то попробуйте следующее:

```
InValidateRect(Form1.Handle,nil,true);
```

// (или взамен передать дескриптор компонента)

// Это очистит холст:

```
Canvas.FillRect(Canvas.ClipRect);
```

// или

```
// Canvas.Brush.Color:=ClWhite;
```

```
// Canvas.FillRect(Canvas.ClipRect);
```

// Самый быстрый способ очистить Canvas

```
PatBlt(Form1.Canvas.Handle, 0, 0,
```

```
Form1.ClientWidth, Form1.ClientHeight, WHITENESS);
```