Московский государственный технический университет
имени Н. Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети

«УТВЕРЖДАЮ»
Заведующий кафедрой ИУ-6
_________ Сюзев В.В.

Г. С. Иванова, Т.Н. Ничушкина

Интерфейсные компоненты Visual Components Library

Методические указания по выполнению лабораторных работ
и домашних заданий в среде Turbo DELPHI 2006

Москва 2011

Оглавление

1 Введение....3
1.1 Типы файлов, используемые Delphi....3
1.2 Основные принципы событийного программирования....4
2 Создание приложений в среде программирования Turbo Delphi.....4
2.1 Создание основного окна проекта....5
2.2 Основные свойства и методы формы (класс TForm)....6
3 Основные свойства и методы простейших компонентов интерфейса....7
3.1 Метка (класс TLabel)....8
3.2 Строчный редактор (класс TEdit).....8
3.3 Текстовый редактор (класс TMemo)....10
3.4 Кнопка (класс TButton)....10
3.5 Кнопка с графикой (TBitBtn)....11
3.6 Кнопка выбора опции (класс TCheckBox)....12
3.7 Радиокнопка (класс TRadioBotton)....12
3.8 Кнопка с фиксацией (класс TSpeedButton)....12
3.9 Список (класс TListBox)....13
3.10 Раскрывающийся список (класс TComboBox)....14
3.11 Панель кнопок выбора (класс TGroupBox)....14
3.12 Группа радиокнопок (класс TRadioGroup)....14
3.13 Панель (класс TPanel)....15
3.14 Кнопка-счетчик (класс TUpDown)....15
3.15 Окно редактирования со счетчиком (класс TSpinEdit)....16
3.16 Изображение (Класс TImage)....16
4 Компоненты TCustomGrid, TDrawGrid, TStringGrid и ListView....17
4.1 Класс TCustomGrid....17
4.2 Компонент-таблица TDrawGrid....20
4.3 Таблица StringGrid....22
4.4 Компонент TListView....27
5 Построение и обработка графических изображений....29
5.1 Обработка событий мыши....29
5.1.1 Событие "Нажатие клавиши мыши"....29
5.1.2 Событие "Движение мыши"....30
5.1.3 Событие "Отпускание клавиши мыши"....30
5.2 Создание графических изображений в среде Turbo Delphi....30
5.2.1 Компонент перо (класс TPen)....30
5.2.2 Компонент Кисть (класс TBrush)....31
5.2.3 Компонент Шрифт (класс TFont)....31
5.2.4 Компонент Канва (класс TCanvas)....32
5.3 Пример построения графического изображения....33

1 Введение

Интегрированная среда программирования Turbo Delphi предназначена для создания 32^{x} разрядных приложений WINDOWS. Эта среда является частью профессиональной среды программирования Delphi Studio (2006 г.) и относится к классу визуальных, в которых разработчику предоставляется возможность прямо на экране формировать интерфейс разрабатываемого программного продукта из стандартных элементов управления.

Языком программирования для среды Turbo Delphi является язык *Object Pascal*, являющийся дальнейшим развитием Borland/Turbo Pascal.

1.1 Типы файлов, используемые Delphi

Среда Turbo Delphi предназначена для создания больших программ, элементы которых размещаются в разных файлах. Среда позволяет создавать проекты, модули форм, модули разработчика, библиотеки DLL, а также текстовые файлы.

Основной частью программы является *проект*. Среда Turbo Delphi создает два файла программы, содержащие проект, которые имеют расширение ***.bdsproj*** (Borland Developer Studio Project File) и ***.dpr*** (Delphi Project File). Файл с расширением ***.dpr*** создается в формате, совместимом с ранними версиями Delphi (в частности, с Delphi 7). Файл с расширением ***.bdsproj*** формируется в формате Delphi Studio. При работе в среде Turbo Delphi файлы равноправны. Запуск любого из них вызовет открытие проекта. Если сформированный в среде Turbo Delphi проект необходимо запустить в среде Delphi 7, следует работать с файлом ***.dpr.*** Как правило, эта часть программы генерируется самой средой Turbo Delphi, но при необходимости разработчик может ее изменить.

Помимо проекта программа может включать различные *модули* (Unit**)**, которые содержатся в файлах с расширением ***.pas***. Часть модулей, как обычно, стандартны и содержат процедуры и функции, выполняющие операции ввода-вывода и т. п., а остальные – добавляются разработчиком при написании программы. Среди добавляемых модулей принято различать модули, содержащие информацию о *формах*, и модули, которые содержат процедуры и функции, непосредственно связанные с решением задачи.

Кроме указанных компонентов программа может использовать динамически подключаемые библиотеки DLL, файлы которых имеют расширение ***.dll***.

При создании программы используется также библиотека стандартных компонентов DCL (файлы которых имеют расширение ***.dcl***), содержащая особым образом подготовленные классы.

После успешной компиляции программы создается исполняемый файл с именем, совпадающим с именем проекта, и расширением ***.exe***, а также файлы – результаты компиляции модулей с расширением ***.dcu***.

Помимо указанных файлов при работе в Turbo Delphi формируются файлы ресурсов с расширением ***.res***, файлы конфигурации с расширением ***.cfg*** для проекта и с расширением ***.dfm*** для модулей форм. В эти файлы помещаются параметры проекта и его компонентов, а также графические изображения, заданные в результате визуального программирования. При этом если в процессе разработки эти файлы случайно потеряются, то файл ***.res*** система предложит пересоздать, а файл ***.cfg*** пересоздаст автоматически. Отсутствие же файла ***.dfm*** приведет к невозможности дальнейшей работы с проектом.

Кроме того, в директории проекта присутствуют файлы с расширением ***.identcache*** и с расширением ***.bdsproject.local***. В них содержится информация о некоторых характеристиках проекта. При отсутствии этих файлов среда пересоздаст их. Таким образом, минимальный набор файлов, необходимый для переноса проекта на другую машину, включает: файл проекта с расширением ***.dpr***, файлы модулей с расширением ***.pas*** и файлы форм с расширением ***.dfm***.

1.2 Основные принципы событийного программирования

Событийным называется программирование, при котором программа представляет собой набор обработчиков некоторых *событий*. В качестве событий при этом могут интерпретироваться: нажатие какой-либо “кнопки” в окне программы, ввод символа с клавиатуры и некоторые ситуации в самой программе (например, открытие или закрытие формы). Таким образом, основной цикл работы программы представляет собой ожидание какого-либо события, вызов соответствующего обработчика для обработки этого события, после чего вновь следует ожидание события.

Основная программа при этом не имеет алгоритма в традиционном смысле, так как связь между отдельными частями не задана жестко, а зависит от последовательности наступления тех или иных событий.

Задача разработчика в этом случае – определить множество событий для программируемой задачи и написать соответствующие обработчики. Причем **Turbo Delphi** предоставляет как стандартные обработчики некоторых событий, так и заготовки для новых, добавляемых обработчиков.

2 Создание приложений в среде программирования Turbo Delphi.

При вызове интегрированной среды **Turbo Delphi** на экране появляется окно, вид которого представлен на рисунке 1.

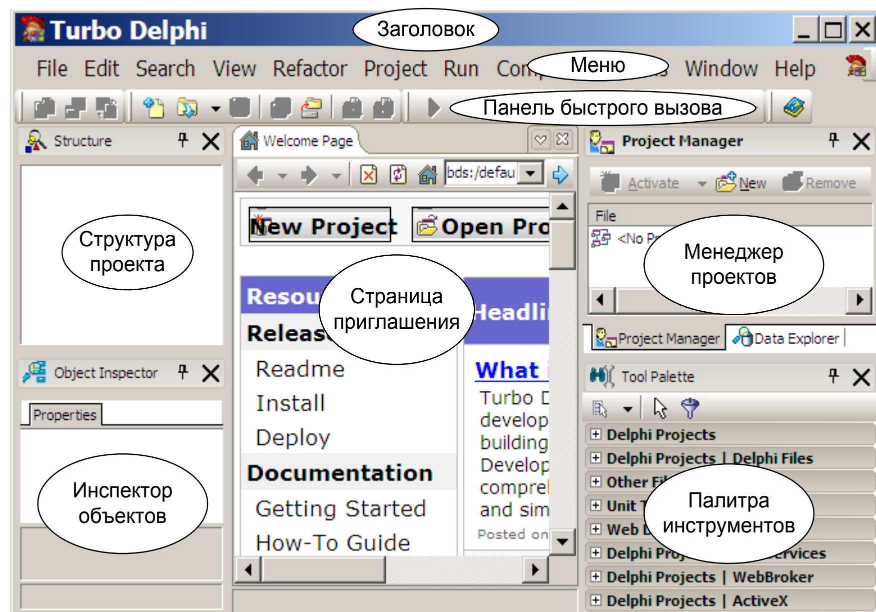


Рисунок 1 – Вид окна Turbo Delphi при входе в среду

Основными элементами данного окна являются:

страница приглашения

главное меню;

панель быстрого доступа;
менеджер проектов (**Project Manager**)
панель структуры проектов (**Structure**)
палитра инструментов (**Tool Palette**);
инспектор объектов (**Object Inspector**);

Работа в среде осуществляется с использованием основного и вспомогательных меню, а также кнопок панели быстрого вызова. Эти кнопки применяют для упрощения доступа к часто выполняемым операциям (они дублируют соответствующие пункты меню).

2.1 Создание основного окна проекта

Разработка программы начинается с создания нового проекта. Это можно сделать двумя способами:

- нажав закладку **New Project** на странице приглашения. После этого на экране появится окно диалога выбора типа проекта, в котором необходимо выбрать тип **VCL Forms Application;**
- выбрав пункт меню **File\ New Project\ VCL Forms Application – Delphi for Win32.**

После этого на экране появится заготовка формы проекта (см. рисунок 2).

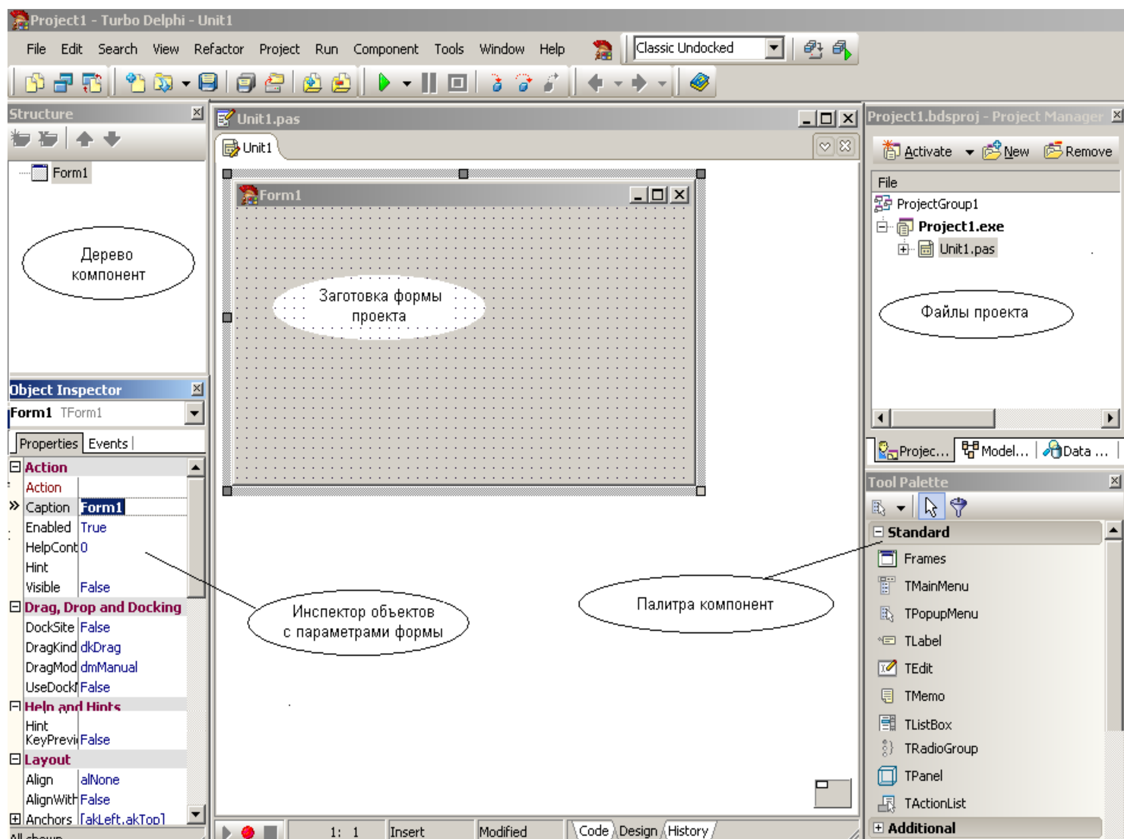


Рисунок 2 – Вид окна приложения с заготовкой формы проекта

Работа над новым проектом начинается с создания основной формы – главного окна приложения. Сначала необходимо настроить форму, затем добавить к форме необходимые *компоненты интерфейса* (поля отображения информации, поля ввода, командные кнопки).

Форма строится на основе класса **TForm** и обладает большим количеством характеристик, как собственных, так и наследуемых от своих предков. Они

предназначены для функционирования формы как окна Windows, а также для выполнения своих специфических задач.

Настройка формы осуществляется путем изменения значений *свойств*. *События*, обрабатываемые формой, определяют ее поведение. Свойства и события можно задавать с помощью *инспектора объектов* (см. рисунок 2).

Для изменения значений свойств используется вкладка **Properties** (свойства). В левой колонке этой вкладки перечислены свойства выбранного объекта (формы, кнопки и т.д.), в правой – указаны значения перечисленных свойств. На вкладке Properties свойства объединены в группы по функциональному признаку, причем названия групп выделены цветом.

Группа **Visual** содержит свойства, определяющие вид объекта (заголовок, цвет фона, вид границы и т.п.).

Группа **Layout** – свойства, определяющие положение объекта (координаты верхнего левого угла, длину, ширину и т. д.).

Некоторые свойства могут отображаться в нескольких группах.

Для определения поведения объекта используется вкладка **Events** (события). Она позволяет задать множество событий, на которые будет реагировать объект. В левой колонке этой вкладки перечислены события, которые может воспринимать объект, а в правой указываются имена обработчиков для выбранных событий.

2.2 Основные свойства и методы формы (класс TForm)

Свойства:

Name – имя формы. Используется для доступа к форме, ее свойствам и методам, а также к компонентам формы.
Caption – текст заголовка (название приложения в заголовке формы).
Width – ширина формы.
Heigth – высота формы.
Position – положение окна в момент старта приложения. Принимает значения:
poCenterScreen – в центре экрана,
poOwnerFormScreen – в центре родительского окна,
poDesigned – положение окна определяют свойства *Top* и *Left.*
Top – расстояние от верхней границы формы до верхней границы экрана.
Left – расстояние от левой границы формы до левой границы экрана.
BorderStyle – вид границы. Может принимать значения:
bsSizeable – обычная, *bsSingle* – тонкая, *bsNone* – отсутствует.
BorderIcons – наличие кнопок управления окном:
biSystemMenu – доступна кнопка системного меню,
biMinimaze – присутствует кнопка Свернуть,
biMaximaze – присутствует кнопка Развернуть,
biHelp – есть кнопка вывода справочной информации.
Icon – значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню.
Color – цвет фона формы.
Font – определяет шрифт текста, выводимого на форму.

Методы:

Show – показать форму.
Hide – спрятать форму.
Close – закрыть форму.

ShowModal – показать форму в модальном режиме. В этом режиме форма всегда остается сверху всех форм своего приложения и открыта, пока пользователь не выберет ответ на заданный вопрос. При закрытии форма возвращает код выбранного ответа.

Обрабатываемые события:

а) *при изменении состояния формы:*

OnCreate – в начальной стадии создания формы – используется при необходимости задания параметров (например, цвета или размера).

OnActivate – при получении формой фокуса ввода, в этом случае окно становится активным и ему адресуется весь ввод с клавиатуры.

OnShow – когда форма (окно) становится видимой.

OnPaint – при необходимости нарисовать или перерисовать форму.

OnResize – при изменении размеров формы на экране.

OnDeactivate – при потере формой фокуса ввода, когда окно становится неактивным.

OnHide – при удалении формы с экрана, когда окно становится невидимым.

OnCloseQuery – при попытке закрыть форму – обычно используется для создания запроса-подтверждения необходимости закрытия окна.

OnClose – при закрытии формы.

OnDestroy – при уничтожении объекта формы.

б) от клавиатуры и мыши:

OnKeyPressed – при нажатии клавиш, которым соответствует код ASCII.

OnKeyDoun, OnKeyUp – при нажатии и отпускании любых клавиш.

OnClick, OnDblClick – при обычном и двойном нажатии клавиш мыши.

OnMouseMove – при перемещении мыши (формируется многократно).

OnMouseDown, OnMouseUp – при нажатии и отпускании клавиш мыши.

в) при перетаскивании объекта мышью:

OnDragDrop – в момент опускания объекта на форму.

OnDragOver – в процессе перетаскивания объекта над формой (многократно).

г) другие:

OnHelp – при вызове подсказки.

OnChange – при изменении содержимого компонентов.

3 Основные свойства и методы простейших компонентов интерфейса

Компонент – это элемент пользовательского интерфейса, обеспечивающий взаимодействие пользователя с приложением. Стандартные компоненты

Delphi объединены в библиотеку, которая называется VCL (Visual Component Library – библиотека визуальных компонентов).

Компоненты, которые программист может использовать в процессе разработки программы, находятся на вкладках палитры компонентов (**Tool Palette**). Часто используемые компоненты находятся на вкладках: **Standard**, **Additional**, **System**.

3.1 Метка (класс TLabel)

Метка представляет собой окно с текстом и может использоваться для расположения на форме некоторых надписей или подписей. Компонент расположен на вкладке Standard.

Свойства:

Caption – заголовок – содержит выводимый в окне компонента текст.

Align – определяет способ выравнивания самого компонента:

alNone – как определено разработчиком;

alTop – занимает всю верхнюю часть окна, в котором размещается;

alBottom – занимает всю нижнюю часть окна, в котором размещается;

alLeft – занимает всю левую часть окна, в котором размещается;

alRight – занимает всю правую часть окна, в котором размещается.

Alignment – определяет способ выравнивания текста в окне компонента:

taCenter – по центру;

taLeftJustify – по левой границе;

taRightJustify – по правой границе.

Autosize – указывает, будет ли размер окна компонента определяться размером введенного текста с учетом шрифта (да, если *true*).

Font – определяет шрифт текста (выбирается в специальном окне).

Color – определяет цвета текста и фона в окне.

WordWrap – определяет, разбивать или нет текст на строки (да, если *true*, при этом значение свойства Autosize должно быть *false*).

Transparent – определяет, виден ли рисунок фона через окно (да, если *true*).

3.2 Строчный редактор (класс TEdit).

Компонент расположен на вкладке Standard и представляет собой окно, обычно выделенное цветом, которое может использоваться, например, для ввода информации.

Свойства:

Text – строка, которая содержит введенную и отображаемую в окне компонента информацию. Доступ к информации в строке осуществляется как к полю записи, например: `Edit1.Text` .

MaxLength – максимальная длина вводимой строки (если равна 0, то длина не ограничена).

ReadOnly – определяет возможность ввода информации в окно компонента (если *true*, то ввод невозможен).

PasswordChar – код символа, который заменяет вводимые символы, например, при вводе пароля (#0 – означает, что отображаются вводимые символы).

AutoSelect – определяет возможность выделения всего текста, отображенного в окне, при фокусировке на данном компоненте при вызове формы; может использоваться, например, если при вводе информации автоматически предлагается какой-либо вариант.

Visible – позволяет скрыть компонент (*false*) или сделать его видимым (*true*).

Методы:

Clear – очистка поля *Text.*

GetTextLen – определить длину строки в поле *Text.*

GetTextBuf – поместить строку в буфер, изменив тип строки (из строки Паскаля сделать строку С).

SetTextBuf – поместить строку из буфера в *Text*, преобразовав строку из строки Си в строку Паскаля.

Например:

```
Procedure TForm1.Button1Click(Sender: TObject);
Var Buffer: PChar;       {тип – указатель на символ – строка Си}
    Size: Byte;
Begin   Size := Edit1.GetTextLen;      {получить длину текста}
        Inc(Size);        {добавить 1 для размещения 0-символа}
        GetMem(Buffer, Size);     {запросить память под буфер}
        Edit1.GetTextBuf(Buffer,Size);{поместить текст в буфер}
        Edit2.Text:=StrPas(Buffer);  {записать текст в Memo2}
        FreeMem(Buffer, Size);          {освободить память}
end;
```

Дополнительно используется несколько параметров и методов, позволяющих работать с выделенной в окне компонента информацией в процессе работы приложения:

Свойства:

SelText – содержит выделенный в окне компонента текст.
SelStart – содержит номер позиции первого выделенного символа.
SelLength – содержит длину выделенного фрагмента.

Методы:

SelectAll – выделить все.
ClearSelection – снять выделение.
CutToClipboard – вырезать выделенную информацию и поместить в буфер.
CopyToClipboard – копировать выделенную информацию и поместить в буфер.
PasteFromClipboard – вставить из буфера.

3.3 Текстовый редактор (класс TMemo)

Компонент расположен на вкладке **Standard.** Представляет собой многострочный текстовый редактор и используется обычно для ввода или отображения текстов.

Свойства:

Text – содержит текст, как единое целое.
Lines – позволяет работать с отдельными строками текста (массив строк типа *TStrings*).

В данном случае применимы методы, работающие со строками: **Add**, **Delete**, **Insert**.

Например:

```
Memo1.Lines.Add('Another line is added');
```

а также методы : **LoadFromFile** и **SaveToFile**. Например:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   Memo1.Lines.LoadFromFile('C:\AUTOEXEC.BAT');
   Writeln('The 6th line of AUTOEXEC.BAT is: ', Memo1.Lines[5]);
end;
```

ReadOnly –определяет возможность ввода информации в окно компонента (если *true*, то ввод невозможен).
MaxLength – определяет максимальную длину текста (если 0, то длина не ограничена).
ScrollBars – определяет наличие или отсутствие полос прокрутки:
ssNone – полосы прокрутки отсутствуют;
ssBoth – обе полосы (горизонтальная и вертикальная);
ssHorizontal – горизонтальная полоса;
ssVertical – вертикальная полоса.
AutoSize – определяет, зависит ли размер окна компонента от параметров текста (да, если *true*).
WordWrap – определяет, выполняется ли автоматическое разбиение строк при выводе в окне компонента (да, если *true*).
WordTabs – определяет, будут ли учитываться символы табуляции при отображении текста (да, если *true*).

Этот компонент наследует все методы, а также свойства и методы, работающие с выделением, описанные для класса **TEdit**.

3.4 Кнопка (класс TButton)

Компонент расположен на вкладке **Standard.** Представляет собой прямоугольник, на котором размещается название кнопки. Используется для инициирования каких-либо действий.

Свойства:

Caption – название кнопки.
Default – определяет, генерируется ли событие *OnClick* для данной кнопки при нажатии клавиши *Enter*, т. е. можно ли пользоваться для «нажатия»

кнопки клавиатурой (да, если *true*). Используется для указания действий, осуществляемых при нажатии клавиши *Enter*.

Cancel – аналогично, но для клавиши *Esc.*

ModalResult – в процессе выполнения в это поле можно занести код возврата, который затем можно проанализировать для определения дальнейших действий.

Visible – позволяет скрыть компонент (*false*) или сделать его видимым (*true*)

Enabled – признак доступности кнопки. Если значение свойства равно *true*, то кнопка доступна, если *false* – то недоступна (в результате щелчка по кнопке, событие *OnClick* не возникает).

Hint – текст подсказки, который появляется рядом с указателем мыши при позиционировании указателя на кнопке; значение свойства ShowHint должно быть *true*.

ShowHint – свойство разрешает (*true*) или запрещает (*false*) отображение подсказки при позиционировании указателя на кнопке.

Основные методы:

OnClick – при обычном нажатии клавиш мыши.

3.5 Кнопка с графикой (TBitBtn)

Компонент расположен на вкладке **Standard.** Представляет собой прямоугольник, на котором размещается битовая графика (например, кнопка OK с галочкой). Используется для инициирования каких-либо действий. Кнопка имеет свойства и события, аналогичные кнопке Button. Однако есть свойства, отражающие особенности кнопки BitBtn.

Свойства:

Glypht – задает изображение на кнопке BitBtn. Чтобы задать битовый образ, надо в окне **Object Inspector** выбрать свойство Glypht, сделать щелчок на кнопке с тремя точками. В появившемся окне **Picture Editor** щелкнуть на кнопке **Load** и в окне **Load Picture** выбрать BMP – файл, в котором находится битовый образ. После нажатия кнопки OK, выбранное изображение появится на кнопке левее надписи. Файл изображения для кнопки может содержать до четырех пиктограмм размера 16*16. Самое левое соответствует отжатой кнопке, второе слева – недоступной кнопке, когда ее свойство **Enabled** равно *false*, третье слева изображение используется при нажатии пользователем на кнопку при ее включении. Четвертое слово используется для кнопки с фиксацией (SpeedButton). Большинство изображений для кнопок используют две пиктограммы.

NumGlyphs – определяет количество картинок в битовом образе Glypht.

Margin – определяет расположение изображения и надписи на кнопке. Если свойство равно 1 (принимается по умолчанию), то изображение и надпись размещаются в центре кнопки. Если свойство больше 0, то в зависимости от значения свойства **Layout**, изображение и надпись смещаются к той или иной кромке кнопки, отступая от нее на число пикселей, указанных в свойстве **Margin**

Layout – определяет положение изображения по отношению к надписи. Может принимать значения: *blGlypLeft* – слева, *blGlypRight* – справа, *blGlypTop* – сверху, *blGlypButton* – снизу.

Spacing – задает число пикселей, разделяющих изображение и надпись на поверхности кнопки. По умолчанию значение свойства равно 4. Если зна-

чение равно 1, то изображение и надпись размещены вплотную друг к другу. Если значение свойства равно 1, то текст появится посередине между изображением и краем кнопки.

Kind – определяет вид кнопки. По умолчанию значение этого свойства равно *bkCustom* – заказная. Но можно установить и множество других предопределенных типов: *bkOK, bkCancel, bkHelp, bkYes,bkNo, bkClose,bkAll* и др. В этих типах уже сделаны соответствующие надписи, пиктограммы и еще некоторые свойства. Обычно все-таки ими лучше не пользоваться. Лучше использовать заказные кнопки и самим задавать в них все необходимые свойства.

3.6 Кнопка выбора опции (класс TCheckBox)

Компонент расположен на вкладке **Standard.** Является независимой кнопкой (флажком) и представляет собой квадратик, внутри которого стоит или не стоит галочка. Используется для обозначения выбора или отмены опций.

Свойства:

Caption – название кнопки (заголовок располагается правее кнопки).
Checked – определяет включенное (*true*) и выключенное (*false*) состояние кнопки.
Alignment – определяет выравнивание заголовка:
taLegtJustify – выравнивание по левой границе окна компонента;
taRightJustify – выравнивание по правой границе.
AllowGrayed – определяет количество состояний кнопки (*true* – 3 состояния, включая неопределенное – серая галочка, а *false* – 2 состояния).
State – определяет состояния кнопки (флажка):
cbGrayed – неопределенное состояние (серая галочка);
cbChecked – включено (флажок установлен);
cbUnchecked – выключено (флажок сброшен).

3.7 Радиокнопка (класс TRadioBotton)

Компонент расположен на вкладке **Standard** и является зависимой кнопкой, состояние которой определяется состоянием других кнопок. Представляет собой кружок, внутри которого стоит или не стоит точка. Используется также как и CheckBox для обозначения выбора или отмены опций, но принято использовать эти кнопки, для взаимоисключающих вариантов.

Все свойства аналогичны кнопке выбора опции CheckBox.

3.8 Кнопка с фиксацией (класс TSpeedButton)

Компонент расположен на вкладке **Standard** и представляет собой командную кнопку, на которой находится картинка. Обычно используется для создания инструментальных панелей (компонент Panel или ToolBar) или в других случаях, когда требуется кнопка с фиксацией нажатого состояния.

Свойства:

Name – имя компонента в описании класса формы.
Glypht – битовый образ, в котором находятся картинки для каждого из возможных состояний кнопки (доступна, недоступна, нажата, зафиксирована)
NumGlyphs – количество картинок в битовом образе Glypht.

Чтобы задать битовый образ, надо в окне **Object Inspector** выбрать свойство Glypht, сделать щелчок на кнопке с тремя точками. В появившемся окне

Picture Editor щелкнуть на кнопке **Load** и в окне **Load Picture** выбрать BMP – файл, в котором находится битовый образ.

Flat – определяет вид кнопки (наличие границ). Если значение свойства равно *true*, то граница кнопки появляется только при позиционировании указателя мыши на кнопке.

GroupIndex – идентификатор группы кнопок. Кнопки, имеющие одинаковый идентификатор группы, работают подобно переключателям (RadioButton): нажатие одной из кнопок группы вызывает срабатывание других кнопок этой группы. Чтобы кнопку можно было зафиксировать, значение свойства не должно быть равно нулю GroupIndex.

Down – идентификатор состояния кнопки. Изменить значение свойства можно, если значение свойства GroupIndex не равно нулю.

AllowAllUp – свойство определяет возможность отжать кнопку. Если кнопка нажата и значение свойства равно *true*, то кнопку можно отжать

Left – расстояние от левой границы кнопки до левой границы формы.

Top – расстояние от верхней границы кнопки до верхней границы формы.

Height – высота кнопки.

Width – ширина кнопки.

Enabled – признак доступности кнопки. Если значение свойства равно *true*, то кнопка доступна, если *false* – то недоступна (в результате щелчка по кнопке, событие *click* не возникает).

Visible – позволяет скрыть компонент (*false*) или сделать его видимым (*true*).

Hint – текст подсказки, который появляется рядом с указателем мыши при позиционировании указателя на кнопке (значение свойства ShowHint должно быть true).

ShowHint – свойство разрешает (true) или запрещает (false) отображение подсказки при позиционировании указателя на кнопке.

Основные методы:

OnClick – при обычном нажатии клавиш мыши. Состояние, которое примет кнопка в результате щелчка на ней, зависит от значения свойства Down. Если значение равно *true*, то кнопка нажата.

3.9 Список (класс TListBox)

Компонент расположен на вкладке **Standard** и является списком, в котором можно выбрать нужный элемент. Используется для отображения вариантов. Представляет собой окно, содержащие отображение некоторых элементов, среди которых можно выбирать.

Свойства:

Items – элементы списка. Представляют собой массив строк типа **TStrings**, выводимых в окно. Для работы с объектами **TStrings** используются методы:

Add – добавить строку к массиву:

```
ListBox1.Items.Add('Новая строка');
```

Delete – удалить строку из массива;

Insert – вставить строку в массив.

Items.count – количество элементов списка.

Sorted – признак автоматической сортировки. Если указано **true**, то строки в окне сортируются.

ItemIndex – номер выбранной строки (строка выделяется). Элементы нумеруются с нуля. Если ни один из элементов не выбран, то значение свойства равно -1.

MultSelect – если указано *true*, то можно выбирать больше, чем одну строку.

ExtendSelect – если указано *true*, то нужно использовать для выбора нескольких строк клавиши **Shift** или **Ctrl**.

SelCount – количество выбранных строк.

Selected – позволяет определить, выбрана ли конкретная строка.

Пример:

```
Procedure TForm1.FormCreate(Sender: TObject);
Var
  I: Integer;
begin
          ListBox1.Items.Add('Blue');
          ListBox1.Items.Add('Yellow');
          ListBox1.Items.Add('Red');
end;
Procedure TForm1.ListBox1Click(Sender: TObject);
Begin
     if ListBox1.Selected[0] then ListBox1.Color:= clBlue;
     if ListBox1.Selected[1] then ListBox1.Color:=clYellow;
     if ListBox1.Selected[2] then ListBox1.Color := clRed;
end;
```

3.10 Раскрывающийся список (класс TComboBox)

Компонент расположен на вкладке **Standard.** Является комбинацией поля редактирования и списка, что дает возможность ввести данные путем набора на клавиатуре или выбором из списка. Представляет собой строку, справа от которой находится кнопка со стрелкой вниз – «выпадающий» список.

Свойства:

Items, ItemIndex, Sorted – аналогичны **ListBox**.

Text – содержит выбранную строку и высвечивается в исходном виде.

3.11 Панель кнопок выбора (класс TGroupBox)

Компонент расположен на вкладке **Standard.** Является контейнером, объединяющим группу связанных органов управления (таких, как RadioBotton, CheckBox и т.д.), и представляет собой панель для размещения кнопок.

Свойства:

Caption – заголовок панели.

3.12 Группа радиокнопок (класс TRadioGroup)

Компонент расположен на вкладке **Standard**. Является комбинацией группового окна GroupBox с набором радиокнопок RadioBotton. Служит специально для создания групп радиокнопок. В компоненте можно размещать несколько радиокнопок.

Свойства:

Items – названия кнопок.
ItemIndex – номер выбранной кнопки.
Columns – определяет размещение кнопок в один или два столбца.

3.13 Панель (класс TPanel)

Компонент расположен на вкладке **Standard.** Является контейнером для группирования органов управления и меньших контейнеров и представляет собой панель для размещения других компонентов. Обычно панель используют для привязки компонентов к границе окна. Также панель можно использовать для построения полос состояния, инструментальных панелей, палитр инструментов.

Свойства:

Align – определяет границу формы, к которой привязана панель и может принимать значения:
alLeft – панель прикреплена к левой границе формы,
alRight – панель прикреплена к правой границе формы,
alTop – панель прикреплена к левой верхней границе,
alBottom – панель прикреплена к нижней границе формы.
BevelOuter – внешняя «фаска» панели:
bvNone – фаска не отображается и поверхность панели находится на одном уровне с поверхностью формы,
bvLowered – поверхность панели притоплена,
bvRaised – поверхность панели выступает над поверхностью формы.
Enabled – свойство позволяет сделать недоступными все компоненты, которые расположены на панели.

3.14 Кнопка-счетчик (класс TUpDown)

Компонент расположен на вкладке **Win32** и представляет собой две соединенные кнопки, используя которые можно уменьшить или увеличить значение внутренней переменной-счетчика. Обычно используется в паре с компонентом Edit, что дает возможность пользователю менять содержимое этого поля.

Свойства:

Position – счетчик. Значение меняется в результате щелчка на кнопке Up (увеличение) или Down (уменьшение).

Min – нижняя граница диапазона изменения счетчика.
Max – верхняя граница диапазона изменения счетчика.
Wrap – определяет поведение компонента при достижении предельных значений (*Min, Max*). Если свойство равно *True*, то попытка превысить максимальное или уменьшить минимальное числа приведет к сбросу счетчика соответственно в минимальное или максимальное значение.
Increment – величина, на которую меняется значение счетчика в результате щелчка на одной из кнопок.
Associate – определяет компонент, используемый в качестве индикатора счетчика. Если используется компонент Edit, то при изменении содержимого поля редактирования, автоматически меняется значение счетчика
AlignButton – определяет, слева или справа от окна будут располагаться кнопки:
udLeft – кнопки расположены слева,

udRight – кнопки расположены справа.

Orientation – задает ориентацию кнопок, может принимать значение:
udHorizontal – кнопки расположены по горизонтали (одна рядом с другой),
udVertical – кнопки расположены по вертикали (одна под другой).

3.15 Окно редактирования со счетчиком (класс TSpinEdit)

Компонент используется для создания окна редактирования в комбинации с кнопкой счетчиком. Свойства компонента похожи на свойства компонента **UpDown**, только имеют другие имена: свойства **Min, Max, Position** называются соответственно **MinValue, MaxValue, Value.** В целом компонент во многих отношениях удобнее простого сочетания **UpDown** и **Edit.** Поэтому, если не нужны дополнительные возможности **UpDown**, то рекомендуется использовать компонент **SpinEdit.**

3.16 Изображение (Класс TImage)

Компонент расположен на вкладке **Additional** и представляет собой некоторую поверхность с канвой, на которую можно заносить изображение. Используется для отображения графической информации, содержащейся в трех видах файлов: пиктограмм, битовых матриц, метафайлов, в которых могут храниться иллюстрации, фотографии, рисунки.

Свойства:

Picture – определяет иллюстрацию, которая отображается в поле компонента. Для выбора изображения необходимо нажать на кнопку с многоточием около свойства *Picture* в окне **Object Inspector.** В появившемся окне **Picture Editor** щелкнуть на кнопке **Load** и в окне **Load Picture** выбрать файл, в котором находится изображение. После нажатия кнопки OK, выбранное изображение появится в компоненте. При этом, компонент не только отображает картинку но и сохраняет в приложении, что позволяет поставлять приложение без отдельного графического файла. Однако в компонент ***Image*** можно загружать внешние файлы и в процессе выполнения приложения.

Width, Height – размеры компонента. Если размер компонента меньше размера рисунка и значение свойств **Autosize, Strech, Proportional** равно *false*, то отображается лишь часть рисунка.

Autosize – определяет возможность автоматического изменения размера компонента в соответствии с реальным размером иллюстрации.

Strech – признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена.

Proportional – признак автоматического масштабирования изображения без искажения. Для выполнения масштабирования значение данного свойства должно быть *true*, а свойство **Autosize –** *false*.

Center – определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно *True*, то картинка располагается в центре поля компонента/

Align – определяет границу формы, к которой «привязан» компонент. Если значение свойства равно alClient, то размер компонента устанавливается

равным размеру «клиентской» (внутренней) области формы, причем, если во время работы программы будет изменен размер формы, автоматически изменен и размер компонента/

Transparent – позволяет сделать изображение прозрачным, если значение данного свойства равно *True*. Используется для наложения изображений. Свойство действует только на битовые матрицы/

Canvas – определяет поверхность компонента.

4 Компоненты TCustomGrid, TDrawGrid, TStringGrid и ListView

4.1 Класс TCustomGrid

При создании приложений часто бывает удобно представлять информацию в виде таблицы. Delphi позволяет создавать различные виды таблиц. Исходным классом всех таблиц является класс ***TCustomGrid*.** Этот класс включает в себя характеристики, общие для любых таблиц, которые можно построить на его основе. Большинство из них определяет внешний вид таблицы.

Form1

Вещественные типы данных

Тип	Диапазан	Число значащих цифр	Размер в байтах
Real	2,9*10-39..1.7*10	11 - 12	6
Singl	1.5*10-45..3.4*10	7 -8	4
Double	5.0*10-324..1.7*1(	15 - 16	8
Extended	3.4*10 -4932..1.1*	19 - 20	10

Фиксированные ячейки

Активная ячейка

Пассивная ячейка

Рисунок 3 – Пример таблицы

Каждая таблица может содержать произвольное количество строк и столбцов. Столбцы и строки могут иметь отображаемые наименования. Размеры отдельных строк и столбцов могут быть произвольными и даже меняться в процессе выполнения программы. В ячейки может помещаться разнообразная текстовая и графическая информация в зависимости от предназначения таблицы. Информацию в ряде случаев можно редактировать.

В таблицах помимо обычных координат, задающих положение и размеры различных элементов в пикселях, существует и другая система координат. В качестве таких координат используют номер столбца и номер строки, в которых расположена ячейка. При этом нумерация строк и столбцов начинается с нуля, например, ячейка с координатами (0,0) является левой верхней ячейкой. Зная размеры ячеек, от этой системы координат можно перейти к традиционной, и наоборот.

Если с ячейкой не выполняются никакие операции, то она находиться в пассивном состоянии. Таблица может содержать, так называемые фиксированные ячейки, которые предназначены для размещения заголовков столбцов и строк. Фиксированные ячейки могут занимать только целые строки и целые столбцы, причем самые верхние и самые левые. Количество фиксированных строк и столбцов можно изменять. Можно, также, задать их цвет. Фиксированные ячейки нельзя выделить, активизировать или отредактировать.

Активная ячейка выделяется либо рамкой из точек, либо специальным цветом. Если предусмотрена возможность редактирования ячеек, то ячейка может дополнительно находиться в редактируемом состоянии.

Ячейки таблицы могут разделяться линиями различной толщины. Имеется возможность изменять во время работы программы ширину столбцов и высоту строк с помощью мыши. Можно также с помощью мыши перемещать строки и столбцы таблицы.

Если вся таблица не умещается в поле, которое для нее отведено, то автоматически появляются линейки скроллинга (если они не запрещены), с помощь которых можно просматривать содержимое таблицы.

Для редактирования текстов используется специальный редактор класса ***TInplaceEditor*.** Этот редактор позволяет изменять содержимое любой ячейки (если редактирование разрешено) за исключением фиксированных ячеек. У редактора есть свойство *Grid*, определяющее то, что он используется совместно с таблицей. *Grid* – это свойство типа *TCustomGrid*, содержащее указатель на таблицу, с которой связан редактор.

Свойства:

ColCount – свойство целого типа, задающее число столбцов.

RowCount – свойство целого типа, задающее число строк таблицы.

Col – свойство целого типа, задающее столбец, в котором находится активная ячейка.

Row – свойство целого типа, задающее строку таблицы, в которой находится активная ячейка.

DefeultColWidth – свойство целого типа задает исходную ширину всех столбцов.

Для индивидуального задания ширины отдельных столбцов используется свойство *ColWidths*.

ColWidths – свойство-массив целого типа с индексами целого типа, задающие ширину каждого столбца.

DefeultRowHeight – свойство целого типа задает исходную высоту всех строк.

Для индивидуального задания высоты отдельных строк используется свойство *RowHeights*.

RowHeights – свойство-массив целого типа с индексами целого типа, задающее высоту каждой строки.

DefeultDrawing – свойство логического типа. Если задано значение *true*, то содержимое ячеек отображается автоматически. Если задано значение *false*, то необходимо создавать свои средства для отображения информации.

FixedCols – свойство целого типа задает число фиксированных столбцов (по умолчанию – один).

FixedRows – свойство целого типа задает число фиксированных строк (по умолчанию – одна).

Примечание. Фиксированные ячейки нельзя редактировать в процессе выполнения программы.

FixedColor – свойство целого типа *TColor* задает цвет фиксированных ячеек.

GridLineWidth – свойство целого типа задает толщину линий между ячейками в пикселах.

LeftCol – свойство целого типа, задающее самый левый столбец таблицы, видимый на экране, не считая фиксированных столбцов.

TopRow – свойство целого типа, задающее самую верхнюю строку таблицы, видимую на экране, не считая фиксированных строк.

VisibleRowCount – свойство целого типа, определяющее число полностью видимых столбцов таблицы на экране, не считая фиксированных.

VisibleColCount – свойство целого типа, определяющее число полностью видимых строк таблицы на экране, не считая фиксированных.

ScrollBars – свойство, определяющее наличие линеек скроллинга.

ssNone – нет линеек,
ssHorizontal – горизонтальная,
ssVertical – вертикальная,
ssBoth – обе линейки.

Selection – свойство, задающее прямоугольник (фактически своими левой верхней и правой нижней ячейками), охватывающий все выделенные ячейки.

Свойства, не отражающиеся в списке, но необходимые для функционирования таблицы.

InplaceEditor – свойство типа *TInplaceEdit* задает встроенный текстовый редактор.

***TCustomGrid*(x, y)** – функция возвращает номер столбца и строки таблицы для координат мыши X и Y (пикселях). Возвращаемый результат имеет тип *TGridCoord*:

```
type
  TGridCoord=record
     X: Longint;
     Y: Longint;
  End;
```

где X – номер столбца таблицы; Y – номер строки таблицы.

TabStops – свойство-массив логического типа с индексами целого типа задает для каждого столбца возможность выделения с помощью клавиши *Tab*.

Options – свойство типа *TGridOptions*, задающее флаги, определяющие поведение таблицы. Тип *TgridOptions* формируется на основе типа *TGridOption*:

type TGridOptions = set of TGridOption;

тип *TGridOption* определяется следующим образом:

```
Type TGridOption=(goFixedVertLine, goFixedHorzline,
         goVertLine, goHorzLine, goRangeSelect,
         goDrawFocusSelected, goRowSizing, goColSizing,
         goRowMoving, goColMoving, goEditing, goTabs,
         goRowSelect,goAlwaysShowEditor,goThumbTracking);
```

где:

goFixedVertLine – фиксированные ячейки разделены вертикальными линиями;
goFixedHorzline – фиксированные ячейки разделены горизонтальными линиями;
goVertLine – остальные ячейки разделяются вертикальными линиями;
goHorzLine – остальные ячейки разделяются горизонтальными линиями;

goRangeSelect – допустимо выделение нескольких ячеек;
goDrawFocusSelected – активная ячейка закрашивается тем же цветом, которым выделяется и выделенная, в противном случае закрашивается цветом нейтральных ячеек;
goRowSizing – высота строк может изменяться;
goColSizing – ширина столбцов таблицы может изменяться;
goRowMoving – строки таблицы могут перемещаться;
goColMoving – столбцы таблицы могут перемещаться;
goEditing – ячейки могут редактироваться;
goTabs – переход от ячейки к ячейке возможен с помощью клавиши *Tab*;
goRowSelect – выделение только целых строк таблицы;
goAlwaysShowEditor – при выделении ячейки она сразу же становится активной (в противном случае должна бать активизирована);
goThumbTracking – перемещение подвижной части таблицы синхронно с перемещением ползунка линейки скроллинга (в противном случае перемещение осуществляется только после того, как ползунок будет отпущен).

4.2 Компонент-таблица TDrawGrid

Компонент в первую очередь предназначен для размещения графической информации (возможно вместе с текстовой информацией). Для объектов класса *TDrawGrid* заданы многие такие же события, что и для других компонентов, например, *OnClick*, *OnEnter*, *OnExit*, *OnKeyPress*, *OnMouseUp* и другие. Вместе с тем, заданы специальные события, возникающие при различных манипуляциях ячейками таблицы.

События:

OnDrawCell – событие, возникающее, когда необходимо перерисовать содержимое ячейки.
OnColumnMoved – событие, возникающее, когда столбец таблицы перемещается с помощью мыши и для свойства *Options* задан флаг *goColMoving*.
OnGetEditMask **–** событие, возникающее, когда необходимо перерисовать содержимое ячейки и для свойства *Options* задан флаг *goEditing.* С помощью этого события задается маска для текста ячейки.
OnGetEditText – событие, возникающее, когда необходимо перерисовать содержимое ячейки и для свойства *Options* задан флаг *goEditing.* С помощью этого события задается исходное значение текста ячейки.
OnRowMoved – событие, возникающее, когда строка таблицы перемещается с помощью мыши и и для свойства *Options* задан флаг *goColMoving.*
OnSetEditText – событие, возникающее после завершения редактирования текста ячейки.

Работа с таблицей – это, в первую очередь, работа с ячейками, следовательно, имеется ряд методов, позволяющих найти размеры тех или иных ячеек или преобразовать одни координаты в другие.

Методы:

CellRect(Col, Row) – функция, определяющая прямоугольник, который занимает ячейка в столбце *Col* и строке *Row*. Используя этот метод, можно сделать ячейку цветной.
MouseToCell(X, Y, Col, Row) – процедура, преобразующая координаты мыши *X* и *Y* в координаты *Col* и *Row* ячейки, на которой находится курсор мыши.

При помощи метода *Selection* можно выделить прямоугольник из ячеек, например оператор:

```
StringGrid1.Selection:= Rect(3,1,2,4);
```

позволяет выделить прямоугольник из четырех ячеек с индексами (3,1), (3,2), (4,1) и (4,2).

Свойство **TopRow** позволяет задать самую верхнюю строку, которая будет видна на экране. Это свойство можно использовать, например, когда таблица не имеет линеек прокрутки, а все строки не помещаются в поле таблицы. В этом случае, используя свойство *TopRow*, можно увидеть невидимые строки, если сделать верхней нужную строку. Например, оператор

`StringGrid1.TopRo :=5;` делает верхней пятую строку.

А если применить оператор

`StringGrid1.TopRo:=StringGrid1.RowCount;` то верхней будет последняя стока таблицы.

Аналогично, для столбцов можно использовать свойство *LeftCol*, которое задает самый правый столбец видимый на экране.

Пример. Создадим таблицу класса *DrawGrid*, в которой в верхней строке будет случайным образом закрашиваться одна из ячеек и затем «падать» вниз.

Для этого разместим на форме компоненту *DrawGrid* из страницы *Additional*. В инспекторе объектов задавим свойству *ColCount* значение 10, а свойству *RowCount* значение12. Теперь наша таблица имеет 10 столбцов и 12 строк. Для того, чтобы избавится от фиксированных строк зададим свойствам *FixedCols* и *FixedRows* значение 0. Изменим размеры таблицы таким образом, чтобы ячейки полностью заняли все ее пространство (чтобы не было линеек скроллинга). Зададим свойству *DefaultDrawing* значение *False*, а свойству *ScrollBars* – значение *ssNone*.

Теперь создадим кнопку *Batton1* и в инспекторе объектов кнопки зададим свойству *Caption* значение «Пуск». На странице *Events* инспектора объектов для кнопки *Batton1* активизируем мышью (двумя щелчками) событие *OnClick* и в заготовку обработчика события поместим следующий текст:

```
procedure TForm1.Button2Click(Sender: TObject);
 var n:integer;   a: array [1..10,1..2] of longint;
   i,j,c,k:longint;  Rect:TRect;
begin
   Randomize;    j:=random(10);
   begin
    Rect:=DrawGrid1.CellRect(j,0); {размеры прямоугольника, за-
                           нимаемого ячейкой с индексами (j,0)}
   c:=Random(clWhite);
   DrawGrid1.Canvas.Brush.Color:=c;     {задание цвета фона}
   DrawGrid1.Canvas.FillRect(Rect);(заполнение прямоугольника
                                                      цветом)
   end;
  {цикл, в котором осуществляется «падение»}
 for i:=1 to 11 do
     begin
      for k:=1 to 100000000 do;          {задержка}
```

```
  {Перерисовка ячеек}
    DrawGrid1.Canvas.Brush.Color:=clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    Rect:=DrawGrid1.CellRect(j,i);
    DrawGrid1.Canvas.Brush.Color:=c;
    DrawGrid1.Canvas.FillRect(Rect);
    end;
end;
```

Прежде, чем заполнить ячейку цветом, мы должны определить координаты прямоугольника, который занимает ячейка. Такие координаты определяются при помощи метода *CellRect(Col, Row)*. В этой процедуре используется переменная *Rect* типа *TRect*. Тип *TRect* имеет следующий вид

```
TRect = record
  case Integer of
    0: (Left, Top, Right, Bottom: Integer);
    1: (TopLeft, BottomRight: TPoint);
end;
```

Координаты прямоугольника могут определяться двумя способами как четыре координаты, задающие координаты в пикселях положение левого верхнего и нижнего правого углов ячейки, или как две ссылки на положение нижний левый и верхний правый угол.

Теперь запустим программу. На экране появится форма с пустой таблицей. Если теперь нажать на кнопку «Пуск», то с верхней строки таблицы начнет падать цветной квадратик (см. рисунок. 4).

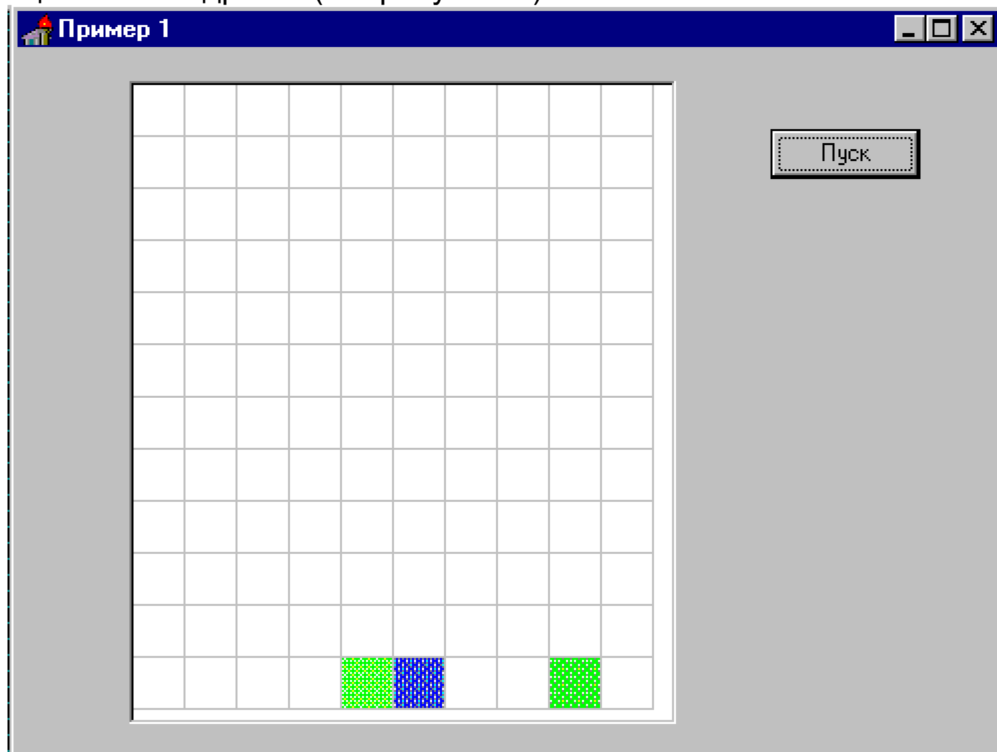


Рисунок 4 – Пример проекта с компонентой DrawGrid

4.3 Таблица StringGrid

Таблица ***TStringGrid*** является потомком класса ***TDrawGrid.*** Таблица класса *TStringGrid* удобна, если нужно работать с текстовой информацией. Также в

этой таблице можно хранить и графические объекты. Кроме наследуемых, класс *TStringGrid* обладает собственными характеристиками.

Свойства:

***Cell*[*Col,Row*] –** свойство-массив типа string. *Col* и *Row* – индексы целого типа, представляющими собой индексы столбцов и строк. Определяет ячейку, находящуюся в столбце *Col* и строке *Row.*

***Cols*[*Index*] –** свойство-массив типа string с индексами целого типа. Определяет столбец таблицы с индексом *Index*.

***Rows*[*Index*] –** свойство-массив типа string с индексами целого типа. Определяет строку таблицы с индексом *Index*.

***Objects*[*Col, Row*] –** свойство-массив типа *TObject* с индексами целого типа. Содержит указатель на объект, находящейся в столбце *Col* и строке *Row*. Например, оператор

```
StringGrid1.Objects[2, 3] := MyIcon;
```

определяет, что в ячейке с индексами (2, 3) будет находиться объект с именем *MyIcon*. Заметим, что сначала нужно создать объект нужного типа. Например, если *MyIcon* имеет тип *TIcon*, то нужно использовать оператор

```
StringGrid1.Objects[2, 3] := TIcon.Create;
```

Наиболее просто работать с такими таблицами в случае, когда ячейки должны содержать только текстовую информацию. В этом случае следует установить свойству *DefoultDrawing* значение *true* и задать для каждой ячейки соответствующий текст, используя свойство *Cell* (это свойство доступно только на этапе выполнения программы).

Пример. Создадим таблицу, в которую занесем фамилии студентов, их экзаменационные оценки по трем предметам и средний балл.

Используя страницу *Additional* палитры компонентов, разместим на форме компонент *StringGrid*. Она получит имя *StringGrid1*. С помощью инспектора объектов удалим горизонтальную линейку прокрутки таблицы строк (потребуется только вертикальная), для чего зададим свойству *ScrollBars* значение *ssVertical*.

Пусть наша таблица будет рассчитана на 20 человек. Зададим свойству *RowCount* значение 21 (одна строка нам понадобится для заголовков столбцов таблицы), а свойству *ColCount* – значение 6. Теперь таблица имеет вид, изображенный на рисунке 5.

В разделе описания типов опишем два новых типа. Тип *s*15 задает строку из 15 символов, которая будет предназначена для хранения фамилии студента. Тип *zap* представляет собой запись, которая состоит из фамилии студента и целочисленного массива для хранения оценок по трем предметам.

```
type  s15=string[15];
    zap= record
      fam: s15;                    {фамилия студента}
      ball:array [1..3] of integer;{массив оценок по трем
                                    предметам}
    end;
```

На странице *Events* инспектора объектов для кнопки *StringGrid1* активизируем мышью (двумя щелчками) событие *OnShow* и в заготовку обработчика события поместим текст программы, при помощи которого информация о студентах, хранящаяся в файле text.txt, разместиться в таблице.

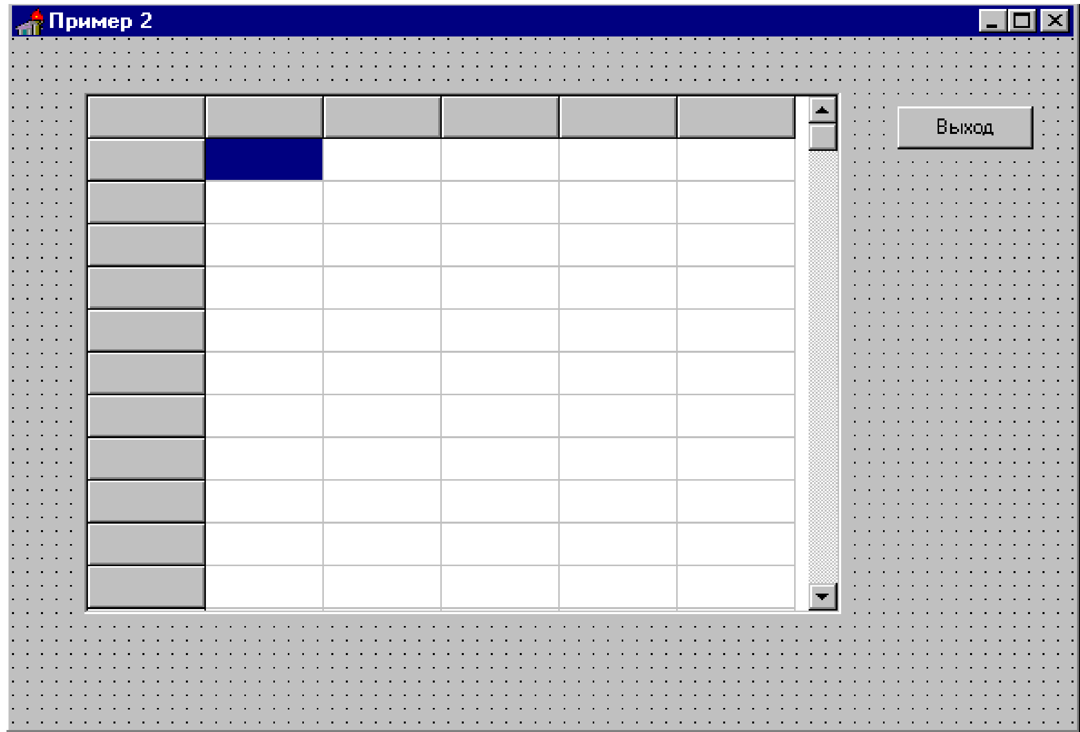


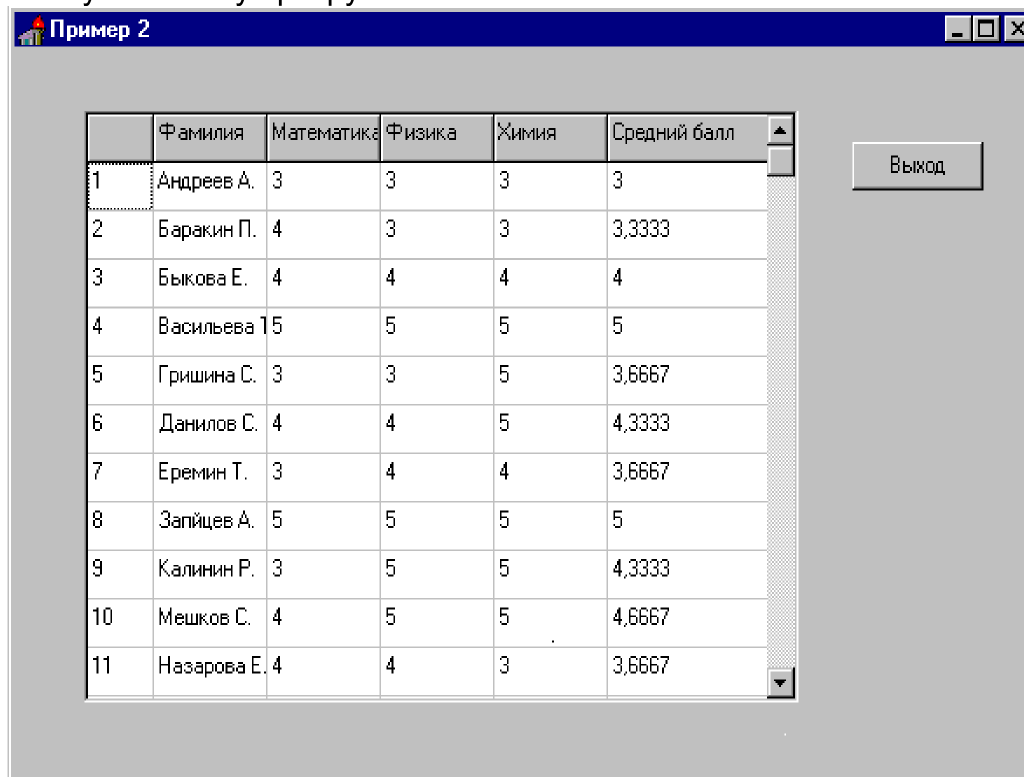
Рисунок 5 – Компонент TStringGrid

Сначала, при помощи метода *Cell* зададим название столбцов таблицы. Затем откроем файл text.txt и, используя цикл, будем читать по одной записи из файла и заносить эти данные в таблицу. Параллельно будет производиться подсчет среднего балла.

```
procedure TForm1.FormShow(Sender: TObject);
var i,j,size:integer;
begin
     {Названия столбцов}
     StringGrid1.Cells[1,0]:='Фамилия';
     StringGrid1.Cells[2,0]:='Математика';
     StringGrid1.Cells[3,0]:='Физика';
     StringGrid1.Cells[4,0]:='Химия';
     StringGrid1.Cells[5,0]:='Средний балл';
     assignFile(f,'text.txt');
     reset(f);
     {заполнение таблицы}
     i:=0;
     while not eof(f)  do
      begin
      i:=i+1;
      StringGrid1.Cells[0,i]:=IntToStr(i);{вывод номера записи}
      read(f,z);                                {чтение записи}
      StringGrid1.Cells[1,i]:=z.fam;     {вывод фамилии}
      mid:=0;
      for j:=1 to 3 do                   {вывод оценок}
       begin
          mid:=Mid+z.ball[j];
          StringGrid1.Cells[j+1,i]:=IntToStr(z.ball[j]);
```

```
      end;
    mid:=mid/3;
    StringGrid1.Cells[5,i]:=FloatToStr(mid);{вывод среднего}
    end;
end;
```

Если теперь запустить проект, то в результате на экране появится таблица, изображенная на рисунке 6. Мы можем просматривать все записи, используя вертикальную линейку прокрутки.



	Фамилия	Математика	Физика	Химия	Средний балл
1	Андреев А.	3	3	3	3
2	Баракин П.	4	3	3	3,3333
3	Быкова Е.	4	4	4	4
4	Васильева Т	5	5	5	5
5	Гришина С.	3	3	5	3,6667
6	Данилов С.	4	4	5	4,3333
7	Еремин Т.	3	4	4	3,6667
8	Запйцев А.	5	5	5	5
9	Калинин Р.	3	5	5	4,3333
10	Мешков С.	4	5	5	4,6667
11	Назарова Е.	4	4	3	3,6667

Рисунок 6 – Пример проекта с компонентом StringGrid

Если кроме текста необходимо отображать изображение, то придется дополнительно создать объекты, содержащие необходимые изображения, поместить указатель на них в свойство *Object* и написать обработчик события *OnDrawCell*, который будет отображать эти изображения.

Заметим, что если значение свойства *DefoultDrawing* имеет значение ***true***, то вначале в верхней части таблице прорисовывается текст, заданный свойством *Cell*, а затем вызывается обработчик события *OnDrawCell*, если этот обработчик назначен. Как правило, обработчик помещает в ячейку изображение, определяемое свойством *Object*. Если свойство *DefoultDrawing* имеет значение *false*, то отображение ячейки полностью определяется обработчиком события *OnDrawCell*.

Пример. Пусть нам нужно задать таблицу, состоящую из трех столбцов и трех строк. В ячейках таблицы должны быть размещены графические изображения. Причем, все изображения одинаковы, кроме одного, которое в начале работы программы находится в центре (см. рисунок 7).

Отличное от других изображение можно переместить, если щелкнуть мышью по другой ячейке.

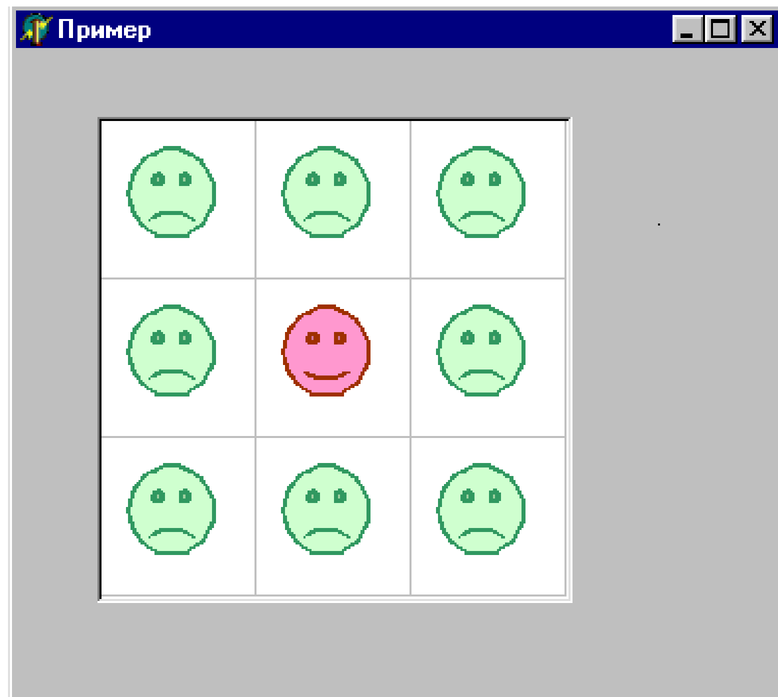


Рисунок 7 – Исходный вид формы

Прежде чем заносить изображение в ячейку таблицы следует сначала создать объект нужного класса. В зависимости от вида изображения, это может быть объект типа *TIcon* для файлов с расширением *ico*, *TBitmap* для файлов с расширением *bmp* и *TMetafile* для файлов с расширением *emf* или *wmf*.

После того, как объект создан, можно загружать графический файл. Например, если мы хотим в ячейке с индексом (1, 1) изобразить картинку, которая сохранена в файле с расширением *bmp*, то можно использовать следующую последовательность команд.

```
Objects[1,1]:=TBitmap.Create;
with    StringGrid1.Objects[1,1] as TBitmap do
                          LoadFromFile('Имя_файла');
```

Для того, чтобы загружать изображение для всех ячеек создадим метод *LoadPic*. Параметрами этого метода являются два целых числа, определяющие ячейку в которой будет находиться отличное от других изображение.

```
procedure TForm1.LoadPic(k,l:integer);
var i,j:integer;
  begin
     with StringGrid1 do
        begin
          for i:=0 to 2 do
           for j:=0 to 2 do
            begin
              Objects[i,j]:=TBitmap.Create;{Создание Bitmap}
              with    StringGrid1.Objects[i,j] as TBitmap do
               if (i=k) and (j=l) then LoadFromFile('U.bmp'
               else LoadFromFile('g.bmp');{Загрузка изображения}
            end;
      end;
 end;
```

В методе *LoadPic* для каждой ячейки создается объект типа *TBitmap*, а затем загружается изображение для ячейки с индексами (i,j) из файла *u.bmp*, а для всех остальных из файла *g.bmp*.

Теперь нужно написать обработчик события *OnDrawCell*, который будет состоять только из одного оператора, вызывающего метод *StringGrid1DrawCell*, который размешает изображение в ячейках. Параметры метода – это координаты левого верхнего угла изображения относительно ячейки и объект, который должен быть изображен. Активизируем мышью событие *OnDrawCell* для таблицы *StringGrid1* на странице *Events* инспектора объектов и запишем следующий текст.

```
procedure TForm1.StringGrid1DrawCell(Sender: TObject; ACol,
      ARow: Integer;  Rect: TRect; State: TGridDrawState);
begin
    StringGrid1.Canvas.Draw(Rect.Left+12,Rect.Top+12,
    TBitmap(StringGrid1.Objects[ACol,ARow]));
end;
```

Выделим форму и в странице *Events* инспектора объектов активизируем мышью событие *OnActivate* формы. Здесь вызовем метод *LoadPic* с параметрами (1,1) для того, чтобы задать начальное изображение таблицы (отличная картинка будет находиться в ячейке с индексами (1,1)).

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  LoadPic(1,1);
end;
```

Теперь создадим обработчик события *OnMouseDown.* Сначала нужно определить индексы ячейки, на которой мы щелкнем мышью, затем загрузить нужные файлы для ячеек и, наконец, перерисовать содержимое ячеек. В результате получим следующую процедуру.

```
procedure TForm1.StringGrid1MouseDown(Sender: TObject;
       Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
  var i,j:integer;
  begin
    StringGrid1.MouseToCell(x,y,i,j);{Определяем индексы ячейки}
    LoadPic(i,j);        {Создаем объекты и загружаем файлы}
    {Перерисовываем содержимое ячеек}
    StringGrid1.OnDrawCell:=StringGrid1DrawCell;
end;
```

4.4 Компонент TListView

Компонент *ListView* тоже можно отнести к таблицам, т.к. он тоже позволяет отображать двумерную информацию. Но, в отличие от других, рассмотренных таблиц, этот компонент позволяет отображать информацию несколькими способами.

Свойства:

Items – позволяет добавлять и удалять элементы.
Columns – с помощью этого свойства можно изменять заголовки столбцов.

ColumnClick – если это свойство имеет значение true, то заголовки столбцов ведут себя подобно кнопке и событие *OnColumnClick* происходит, когда пользователь щелкнул на заголовок столбца.

ViewStyle – определяет каким образом отображаются элементы: вертикально, горизонтально или в столбцах с заголовками. Для этого свойства могут быть установлены значения *vsList*, *vsIcon*, *vsReport* или *vsSmallIcon*.

Если свойство имеет значение *vsIcon*, то каждый элемент будет отображаться как полноразмерный значок, имеющий снизу установленную для него метку.

Пользователь может перемещать элементы по полю компоненты. Если установлено значение *vsSmallIcon*, то элементы списка будут отображаться в виде маленьких значков, также имеющих справа , установленную для них метку. При значении *vsList* элементы располагаются в столбец и их нельзя перемещать. Если свойство *ViewStyle* имеет значение *vsReport*, тогда вся информация располагается в виде таблицы.

Рассмотрим некоторые возможности класса *TListView* на примере.

Пример. Пусть требуется создать список сотрудников, состоящий из фамилии, домашнего адреса и домашнего телефона.

Для решения этой задачи на форме разместим компоненту *ListView*. При помощи свойства Items создадим требуемый список сотрудников. Причем фамилии будут основными элементами, а адрес и телефон – под элементами (см. рисунок 8). Используя свойство *Columns*, зададим названия столбцов будущей таблицы.

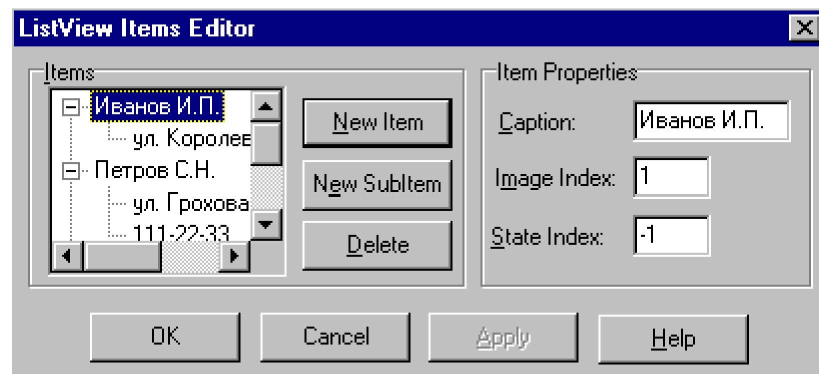


Рисунок 8 – Форма для создания списка элементов компонента ListView

Теперь разместим на форме компонент *RadioGroup*, при помощи которого будем менять вид представления информации. *RagioGroup* будет содержать четыре переключателя, которые назовем: «Мелкие значки, «Крупные значки»», «Список» и «Таблица».

Для события *OnClick* компоненты *RadioGroup* напишем обработчик, который будет изменять вид преставления при изменении активного переключателя.

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  case RadioGroup1.ItemIndex of
     0: ListView1.ViewStyle:=vsSmallIcon;
     1: ListView1.ViewStyle:=vsIcon;
     2: ListView1.ViewStyle:=vsList;
     3: ListView1.ViewStyle:=vsReport;
  end;
end;
```

Теперь, если запустить проект, то в поле компоненты *ListView*, мы увидим список сотрудников. При помощи переключателей мы получаем разный вид информации (см. рисунок. 9).

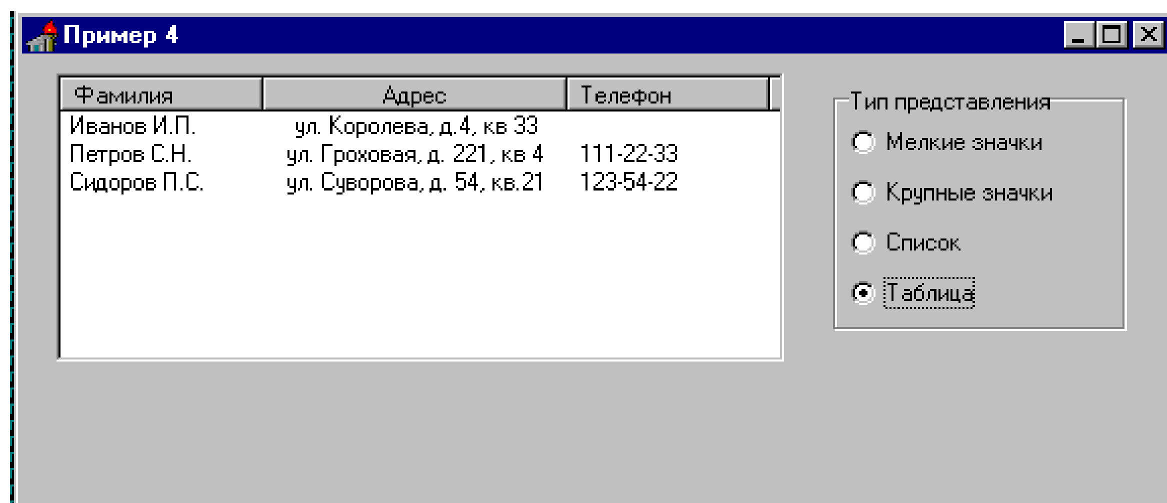


Рисунок 9 – Пример проекта с компонентом ListView.

5 Построение и обработка графических изображений

5.1 Обработка событий мыши

При изменении состояния мыши формируется три события.

5.1.1 Событие "Нажатие клавиши мыши"

Если нажать кнопку мыши над некоторым компонентом, то обработка будет выполняться следующим образом. Окно формы, над которым была нажата кнопка мыши, получит сообщение от мыши. Соответствующий метод-обработчик сообщения, получив это сообщение, определит компонент, над которым была нажата кнопка и, при наличии соответствующего метода-обработчика события передаст ему управление. Заголовок метода-обработчика события имеет следующий формат:

```
procedure <имя компонента>MouseDown(Sender: TObject;
 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
```

где

Button : TMouseButton – параметр, определяющий, какая кнопка нажата:

type **TMouseButton** = (mbLeft, mbRight, mbMiddle); – соответственно, левая, правая или средняя кнопки;

Shift: TShiftState – параметр, определяющий нажатие управляющих клавиш клавиатуры и мыши (одновременно могут быть нажаты клавиши клавиатуры и мыши):

type **TShiftState** = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);

X, Y: Integer – параметры, определяющие координаты мыши относительно компонента.

5.1.2 Событие "Движение мыши"

Если мышь движется с нажатой клавишей над компонентом, то многократно фиксируется событие движения мыши. Заголовок метода-обработчика события имеет следующий формат:

```
procedure <имя компонента>MouseMove(Sender: TObject; Shift:
                               TShiftState; X, Y: Integer);
```

5.1.3 Событие "Отпускание клавиши мыши"

Фиксируется при отпускании клавиши мыши над компонентом. Заголовок метода-обработчика события имеет следующий формат:

```
procedure <имя компонента>MouseUp(Sender:TObject;
    Button:TMouseButton; Shift:TShiftState; X,Y:Integer);
```

Для обработки любого из этих событий необходимо предусмотреть соответствующий обработчик в классе формы.

5.2 Создание графических изображений в среде Turbo Delphi

Изображения в Windows создаются на компоненте **TImage**, который используется в тех случаях, когда на форму необходимо поместить рисунок (готовый или формируемый в процессе работы). Рисунок формируется с помощью следующих типов и компонентов.

Типов

- точка,
- прямоугольник;

Компонентов:

- перо, которое используется для рисования линий;
- кисть, которая используется для закрашивания замкнутых фигур;
- шрифт, который используется при выводе надписей;
- холст, на котором выполняются изображения.

Точка – тип, позволяющий определить точку на экране:

```
type TPoint = record
  X: Longint;
  Y: Longint;
end;
```

Прямоугольник – тип, используемый для задания прямоугольника:

```
type   TRect = record
  case Integer of
    0: (Left, Top, Right, Bottom: Integer);
    1: (TopLeft, BottomRight: TPoint);
end;
```

5.2.1 Компонент перо (класс TPen)

С помощью класса TPen создается объект Перо, служащий для вычерчивания линий, контуров и т. п.

Свойства:

Color:TColor – цвет вычерчиваемых линий;
Width:Integer – толщина линии в пикселях экрана;
Style:TPenStyle – стиль линий – учитывается только для толщины 1 пиксель:
psSolid, psDash, psDot, psDashDot, psDashDotDot, psClear, psInsidFrame;
Mode:TPenMode – способ взаимодействия линий с фоном, например,
pmBlack – только черные линии,
pmWhite – только белые линии,
pmNop – линии не видны на фоне,
pmNot – инверсия фона и т.д..

5.2.2 Компонент Кисть (класс TBrush)

Объекты класса TBrush служат для заполнения внутренего пространства (установки цвета фона и образца заполнения)

Свойства:

Color:TColor – цвет кисти:
clAqua (прозрачный), clBlack, clBlue, clDkGray, clFuchsia, clGray, clGreen, clLime (салатовый), clLtGray, clMaroon (каштановый), clNavy (синий), clOlive, clPurple (фиолетовый), clRed, clSilver (серебряный), clTeal, clWhite, clYellow
Style:TBrushStyle – стиль кисти (образец заполнения), например:
bsSolid (сплошное), bsClear (отсутствующее), bsHorizontal (горизонтальными линиями), bsVertical (вертикальными линиями), bsFDiagonal (прямыми диагоналями), bsBDiagonal (обратными диагоналями), bsCross (крестиками), bsDiagCross (диагональными крестиками);
BitMap:TBitMap – растровое изображение, которое будет использоваться кистью для заполнения, если свойство определено, то цвет и стиль игнорируются.

5.2.3 Компонент Шрифт (класс TFont)

Объект класса TFont определяет шрифт, которым выводится текст.

Свойства:

Charset:TFontCharSet – набор символов:
RUSSIANCHARSET – русский, OEM_CHARSET – текст MS DOS;
Name:TFontName – имя шрифта, по умолчанию – MS Sans Serif;
Color:TColor – цвет;
Height:Integer – высота в пикселях;
Size:Integer – высота в пунктах (1/7 дюйма);
Pitch:TFontPitch – способ расположения букв в тексте:
fpFixed – моноширный текст,
fpVariable – пропорциональный текст,
fpDefault – ширина шрифта по умолчанию;
Style:TFontStyle – стиль шрифта – комбинация из:
fsBold – полужирный, fsItalic – курсив, fsUnderline – подчеркнутый, fsStrikeOut – перечеркнутый.

5.2.4 Компонент Канва (класс TCanvas)

Класс создает Канву – холст для рисования (ПОВЕРХНОСТЬ ДЛЯ РИСОВАНИЯ). Рисование выполняется с помощью свойств и методов, входящих в класс TCanvas.

Свойства:

Brush:TBrush – кисть;
Pen:TPen – перо;
Font:TFont – шрифт;
PenPos:TPoint – определяет текущее положение пера над холстом в пикселях относительно левого верхнего угла;
CopyMode:TCopyMode – способ взаимодействия растрового изображения с цветом фона, используется при копировании части канвы на другую методом CopyRect:
cmBlackness – заполнение черным цветом,
cmDestInvert – заполнение инверсным фоном,
cmSrcCopy – копирует изображение источника на канву и т.д.;
Pixels[X,Y:Integer]:TColor – массив пикселей канвы.

Методы:

procedure MoveTo(X,Y:Integer) – перемещает перо в указанную точку;
procedure LineTo(X,Y:Integer) – чертит линию из текущей точки в заданную;
procedure Rectangle(X1, Y1, X2, Y2: Integer) – рисует и закрашивает кистью прямоугольник;
procedure Polyline(Points:array of TPoint) – рисует ломаную линию;
procedure Polygon(Points: array of TPoint) – рисует и закрашивает кистью многоугольник;
procedure Ellipse(X1, Y1, X2, Y2: Integer)– рисует эллипс в заданном прямоугольнике и закрашивает кистью;
procedure FrameRect(const Rect:TRect) – очерчивает границы прямоугольника текущей кистью без заполнения;
procedure Arc(X1,Y1,X2,Y2,X3,Y3, X4,Y4:integer) – чертит дугу эллипса в прямоугольнике (X1,Y1,X2,Y2), направление – против часовой стрелки;
procedure Chord(X1,Y1,X2,Y2,X3,Y3, X4,Y4:integer) – чертит сегмент эллипса в прямоугольнике (X1,Y1,X2,Y2), направление – против часовой стрелки;
procedure Pie(X1,Y1,X2,Y2,X3,Y3, X4,Y4:integer) – чертит сектор эллипса в прямоугольнике (X1,Y1,X2,Y2), направление – против часовой стрелки;
procedure RoundRect(X1,Y1,X2,Y2, X3,Y3:integer) – чертит и заполняет прямоугольник с закругленными краями
procedure FillRect(const Rect: TRect)– закрашивает кистью прямоугольник, включая левую и верхнюю границы.

```
type TRect = record
case Integer of
      0: (Left, Top, Right, Bottom: Integer);
      1: (TopLeft, BottomRight: TPoint);
end;
```

procedure FloodFill(X,Y:Integer;Color:TColor;FillStyle:TFillStyle) – заливка области, зависит от типа **TFillStyle.**

type TFillStyle = (**fsSurface,fsBorder**);

- **FillStyle=fsBorder -** заливка области с границей цвета **Color;**
- **FillStyle=fsSurface** - заливка области цвета **Color** цветом, определенным кистью.

procedure TextOut(X,Y:Integer; const Text:string)– вывод строки текста шрифтом **TFont** в прямоугольник с верхним левым. углом в точке (X,Y);

function TextExtent(Const Text:String):TSize – возвращает ширину и высоту прямоугольника, охватывающего текстовую строку **Text**;

function TextWidth(Const Text:string):Integer – возвращает ширину прямоугольника, охватывающего текстовую строку;

5.3 Пример построения графического изображения

В качестве примера рассмотрим реализацию фрагмента графического редактора, который рисует на форме закрашенный прямоугольник.

Для рисования закрашенного прямоугольника необходимо отметить первый угол нажатием левой кнопки мыши. Второй угол фиксируется в момент отпускания левой кнопки. Пока кнопка нажата, за курсором мыши должен тянуть контур.

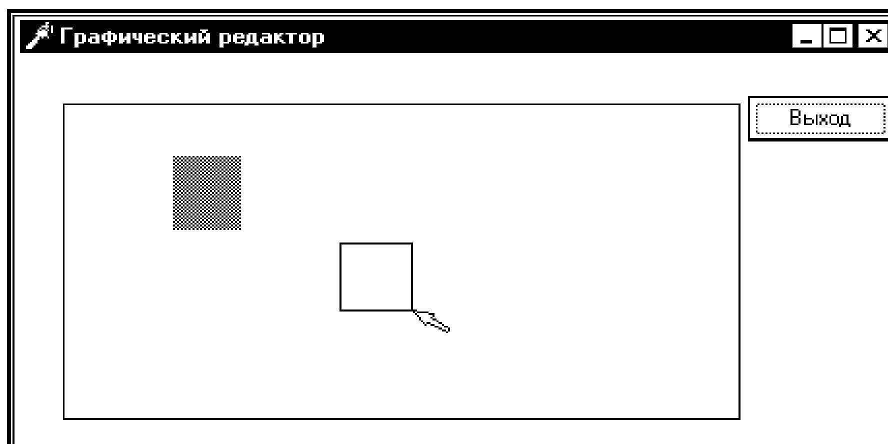


Рисунок 10 – Вид основной формы редактора

```
program Draw;
uses
  Forms,    MainUnit in 'MainUnit.pas' {MainForm};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm (TMainForm, MainForm);
  Application.Run;
end.

unit MainUnit;
interface
uses  Windows, Messages, SysUtils, Classes, Graphics, Controls,
      Forms, Dialogs,  StdCtrls, ExtCtrls;
type
  TMainForm = class(TForm)
    ImagePole: TImage; {поле рисования}
    ExitButton: TButton;  {кнопка выхода}
```

```
    procedure FormActivate(Sender: TObject);
    procedure ImagePoleMouseDown(Sender: TObject;
        Button: TMouseButton;Shift: TShiftState; X, Y: Integer);
         {установка начальной точки}
    procedure ImagePoleMouseMove(Sender: TObject;
        Shift: TShiftState; X, Y:Integer); {тянем контур,
        пока мышь движется с нажатой кнопкой}
    procedure ImagePoleMouseUp(Sender: TObject;
        Button: TMouseButton;Shift: TShiftState; X,Y:Integer );
        {завершаем рисование прямоугольника}
    procedure ExitButtonClick(Sender: TObject);
  end;

var  MainForm: TMainForm;

implementation
   Var Rect:TRect;first:boolean;
{$R *.DFM}

procedure TMainForm.FormActivate(Sender: TObject);
begin ImagePole.Canvas.Brush.Color:=clWhite end;  {Установка
                                               белого цвета фона}
end;

procedure TMainForm.ImagePoleMouseDown(Sender: TObject;
   Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin          {Координаты  первой точки}
   if Button=mbLeft then
      begin     Rect.Left:=x;
                Rect.Top:=y;
               first:=true;
      end;
end;

procedure TMainForm.ImagePoleMouseMove(Sender: TObject;
             Shift: TShiftState; X, Y: Integer);
begin
 if ssLeft in Shift then
  begin
      if first then first:=not first
         else
           begin
            ImagePole.Canvas.Pen.Color:=clWhite;   {Белое перо}
            ImagePole.Canvas.Rectangle(Rect.Left,Rect.Top,
                          Rect.Right,Rect.Bottom);{Стираем}
           end;
     Rect.Right:=X;  {Меняем координаты}
     Rect.Bottom:=Y;
     ImagePole.Canvas.Pen.Color:=clBlack; {Черное перо}
     ImagePole.Canvas.Rectangle(Rect.Left,Rect.Top,
                          Rect.Right,Rect.Bottom);{Рисуем}
  end;
end;
```

```
procedure TMainForm.ImagePoleMouseUp(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X,Y:Integer);
begin
  if Button=mbLeft then
        begin
          ImagePole.Canvas.Pen.Color:=clWhite; {Белое перо}
          ImagePole.Canvas.Rectangle(Rect.Left,Rect.Top,
                Rect.Right,Rect.Bottom); {Стираем}
          Rect.Right:=X; {Меняем координаты}
          Rect.Bottom:=Y;
          ImagePole.Canvas.Brush.Color:=clRed; {Красная кисть}
          {Рисуем последний вариант - красный прямоугольник}
          ImagePole.Canvas.FillRect(Rect);
          ImagePole.Canvas.Pen.Color:=clBlack; {Черное перо}
       end;
end;

procedure TMainForm.ExitButtonClick(Sender: TObject);
begin
   Close;
end;

end.
```