ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РЫБИНСКИЙ ГОСУДАРСТВЕННЫЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ П. А. СОЛОВЬЕВА»
(РГАТУ имени П.А. Соловьева)

ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА ПОДГОТОВКИ АСПИРАНТОВ

направление подготовки 09.06.01 Информатика и вычислительная техника

профиль подготовки 05.13.06 Автоматизация и управление технологическими процессами и производствами (в промышленности)

# ПРАКТИКУМ

**по дисциплине**

## Организация программного обеспечения АСУ

Разработал: д.т.н. Юдин А. В.

Рыбинск, 2014 г.

**Лабораторная работа №1**

**Цель лабораторной работы:** Изучение основ объектно-ориентированного программирования.

*1.1 Объекты и классы*

Класс - это тип объекта. Он характеризует объект вместе с его свойствами и правилами поведения. Объект есть экземпляр класса. В Delphi все объекты динамические. Для выделения памяти под объект используется специальный метод данного класса Constructor Create. Освобождение памяти, выделенной ранее объекту, осуществляется методом Dectructor Destroy. Для объектов и классов сделано одно исключение из общих правил работы на Паскале с динамическими переменными – для них не используется символ ^ (тильда) для указателей и содержимого указываемой памяти. Переменные, описанные в классе, называют полями. Любая процедура или функция, описанная в классе, является уже методом. При вызове любого метода ему неявным образом первым параметром передается параметр Self, который является указателем на объект, который вызвал данный метод. Если процедура описана вне класса, но за ее описанием следуют слова of object, то это тоже будет метод. Классы могут быть описаны или в интерфейсной части модуля Unit или в самом начале секции реализации Implementation. Не допускается их описание внутри процедур и функций. Если перед описанием метода стоит ключевое слово class, то это классовый метод и его можно вызывать даже в том случае, если объекту еще не была выделена память. Типичный пример класса:

```
Type TmyObject=class(Tobject)
x,y:integer;
Constructor Create;
Destructor Destroy;virtual;
Procedure Show;
   End;
```

В скобках после ключевого слова class указывается наследуемый класс. Объект Tobject является прародителем всех классов и не наследует никаких других классов.

*1.2 Области видимости класса*

1. Private – личная, внутренняя область класса. Поля и методы, описанные в этой области, доступны только внутри модуля Unit, где описан данный класс. Для всех других модулей, которые подсоединяют данный модуль и наследуют этот класс, они недоступны.
2. Protected – защищенная область. Поля и методы этой области доступны только внутри классов, наследующих данный класс.
3. Public – общедоступная область. Поля и методы этой области не имеют ограничений на видимость.
4. Published – область публикаций. Поля и методы этой области имеют такую же видимость, как для области public, но они еще видны инспектору объектов на этапе разработки программы. В дочерних классах можно переносить методы и свойства из области Protected в область Published и обратно.

*1.3 Свойства (Property) и инкапсуляция*

Объектно - ориентированное программирование (ООП) основано на трех принципах – инкапсуляция, наследование и полиморфизм. Классическое ООП утверждает, что чтение и обновление полей должно производиться только специальными методами и не допускается прямое обращение к полям класса. Это правило и называется инкапсуляцией, а такие поля - свойствами. Свойство определяется полем и двумя методами, которые осуществляют чтение и запись заданных значений в поле. Пример определения свойства:

```
Type TmyObject=class(Tobject)
Privete
FmyField:String;
Protected
Procedure SetMyField(Value:String);
```

```
Published
Property MyProp:String Read FmyField
Write SetMyField
Default ‘Начальное значение’;
End;
```

Здесь в классе TmyObject определено свойство MyProp строкового типа. В качестве метода чтения выступает само значение строки, а запись осуществляется методом SetMyField. Само поле FmyField определено в области Privete и к нему поэтому нет прямого доступа из других модулей. Метод чтения этого поля находится в защищенной области, а свойство MyProp - в области публикаций и доступно инспектору объектов во время проектирования программы.

### *1.4 Методы, наследование и полиморфизм*

Наследование означает, что при создании нового класса он наследует все поля, свойства и методы, определенные в родительском классе. В новом классе только добавляются новые поля, методы и свойства. Унаследованные от предка поля и методы доступны в дочернем классе, но с учетом областей видимости. Если имеет место совпадение имен, то говорят, что они перекрываются. В Delphi допускается только последовательное единичное наследование классов. Методы подразделяются на 4 группы:

статические (Static),
виртуальные (Virtual),
динамические (Dynamic) и
абстрактные (Abstract).

Адрес вызова статического метода определяется на этапе трансляции проекта и вызов этих методов осуществляется быстрее всех остальных методов. Такие методы можно без ограничений перекрывать, при этом можно менять список передаваемых параметров. По умолчанию методы объектов являются статическими.

Адреса виртуальных и динамических методов определяются во время выполнения программы и находятся в специальных таблицах: таблице виртуальных методов (VMT) и таблице динамических методов (DMT). В таблицу VMT включаются адреса всех определенных в данном классе виртуальных методов и всех наследуемых методов. В таблицу DMT включаются адреса динамических методов определенных только в данном классе. Поэтому виртуальные методы вызываются быстрее динамических, но размеры таблиц VMT существенно больше таблиц DMT. Динамические методы позволяют экономить память, но их вызов осуществляется медленнее всех остальных методов, т.к. приходится для поиска адреса метода проходить по всем таблицам DMT родительских классов пока не найдем нужный нам динамический метод.

Для перекрытия виртуальных и динамических методов используется ключевое слово Override. Список параметров перекрываемых виртуальных и динамических методов не должен отличаться от списка параметров этих методов в родительском классе.

Абстрактные методы определяют только интерфейсную часть метода, такие методы нельзя использовать без перекрытия в дочерних классах, где должна находиться и реализация такого метода.

Рассмотрим следующий пример.

```
Type Tpoint=Class(Tobject)
Constructor Create;
        Destructor Destroy;virtual;
X,y:Integer;
        C:Tcolor;
Procedure Show;virtual;
End;
Tcircle=Class(Tpoint)
Constructor Create;
```

```
Destructor Destroy;override;
R:Integer;
Procedure Show;override;
End;
……..
Var Point1:Tpoint;
Circle1:Tcircle;
Begin
Point1.Create;
Circle1.Create;
With Point1 do Begin
X:=100;y:=50; C:=clRed;
Show;
                    End;
With Circle1 do Begin
X:=200;Y:=100:C:=clBlue;
R:=50;
Show;
End;
……
```

В данном примере определены два класса Tpoint и Tcircle. Класс Tpoint наследует класс Tobject. В классе Tpoint определены поля X, Y, которые задают точку на экране дисплея, и C – цвет этой точки. В нем также описаны методы по выделению и освобождению памяти под объект и виртуальный метод Show - рисования точки на экране. Класс Tcircle наследует класс Tpoint и все его поля и методы. В нем дополнительно описаны поле R (радиус) и метод Show, который перекрывает аналогичный метод класса Tpoint. Метод Show класса Tcircle рисует уже окружность. В результате два метода Show рисуют разные картинки, в зависимости от того, какому классу они принадлежат. Это и называется полиморфизмом объектов.

### *1.5 События (Events)*

События в Delphi - это свойства процедурного типа, предназначенные для создания пользовательской реакции на те или иные входные воздействия. Пример объявления события.

```
Property OnMyEvent:TmyEvent Read FOnMyEvent
Write FonMyEvent;
```

Присвоить такому свойству значение – это значит указать адрес метода, который будет вызываться в момент наступления события. Такие методы называются обработчиками событий. События имеют разные типы, но общим для всех является параметр Sender – указатель на объект источник события. Самый простой тип события это тип

```
TnotifyEvent=procedure(Sender:Tobject) of Object;
```

Здесь «of object» означает, что данный тип определяет именно метод, а не обычную процедуру. Приставка On в имени свойства означает, что данное свойство является событием, хотя не каждое событие может иметь такую приставку. Для определения события необходимо в разделе Private объявить поле указателя на метод. В разделе Protected нужно объявить методы чтения и записи адреса обработчика события в это поле и объявить событие, как свойство процедурного типа. В инспекторе объектов есть две страницы: страница свойств (Properties) и страница событий (Events). Двойной щелчок левой клавишей мыши по событию приводит к появлению обрамления обработчика события в тексте программного модуля Unit.

Далее приведем основной список событий. Общими для всех компонентов являются события (наследники класса TControl):

OnClick – нажатие левой клавиши мыши,

OnDblClick – двойной щелчок левой клавиши мыши,
OnMouseDown – нажатие любой клавиши мыши,
OnMouseMove – перемещение курсора мыши по компоненту,
OnMouseUp – отжатие кнопки мышки.
Общими для оконных элементов управления являются события (наследники класса (TWinControl):
OnEnter – перемещение фокуса ввода на компонент, данный компонент становится активным,
OnExit – потеря активности компонентом,
OnKeyDown – нажатие клавиши или комбинации клавиш,
OnKeyPress – нажатие каждой одиночной клавиши,
On KeyUp – отпускание клавиши.

**Вопросы**

1. Чем класс отличается от объекта?
2. Объекты бывают статическими или динамическими?
3. Для каких целей используется метод Create?
4. Что собой представляет неявно передаваемый в объект параметр Self?
5. Области видимости класса.
6. Что такое свойства объектов?
7. Что обозначает принцип инкапсуляции в ООП?
8. Чем метод отличается от обычной процедуры?
9. Какие вы знаете типы методов?
10. Что означает принцип наследования классов?
11. Что такое полиморфизм в ООП?
12. Что такое событие и чем оно отличается от свойства класса?
13. Приведите примеры основных событий компонентов?
14. Чем динамические методы отличаются от виртуальных?
15. Где можно давать определение классу?

**Лабораторная работа №2**

Основы объектно-ориентированного визуального программирования: графический интерфейс и событийные процедуры.

**Цель лабораторной работы:** научиться работать с графическими объектами.

*3.1 Класс Tcanvas*

Основу графики в Delphi представляет класс Tcanvas. Это холст (контекст GDI в Windows) с набором инструментов для рисования. Основные свойства холста:

Property Pen:Tpen; - карандаш,

Property Brush:Tbrush; - кисть,

Property Font:Tfont; - шрифт,

Property PenPos:Tpoint; - текущая позиция карандаша в пикселях относительно левого верхнего угла канвы,

Property Pixels[x,y:Integer]:Tcolor.

Property CopyMode:TcopyMode; Это свойство определяет, как графический рисунок копируется в канву. Оно используется при вызове метода CopyRect и при копировании объектов TbitMap.

Возможные значения этого свойства приведены ниже.

| Таблица 2 cmBlackness | Заполняет область рисования черным цветом |
|---|---|
| cmDest | Заполняет область рисования цветом фона |
| cmMergeCopy | Объединяет изображение на канве и копируемое изображение операцией AND |
| cmMergePaint | Объединяет изображение на канве и копируемое изображение операцией OR |
| cmNotSrcCopy | Копирует на канву инверсное изображение источника |
| cmNotSrcErase | Объединяет изображение на канве и копируемое изображение операцией OR и инвертирует полученное |
| cmPatCopy | Копирует шаблон источника |
| cmPatInvert | Комбинирует шаблон источника с изображением на канве с помощью операции XOR |
| cmPatPaint | Комбинирует инверсное изображение источника с исходным шаблоном, используя операцию OR. Смешивает результат этого действия с изображением на холсте, используя логическую операцию OR |
| cmSrcAnd | Объединяет изображение источника и канвы с помощью операции AND |
| cmSrcCopy | Копирует изображение источника на канву |
| cmSrcErase | Инвертирует изображение на канве и объединяет результат с изображением источника операцией AND |
| cmSrcInvert | Объединяет изображение на канве и источнике операцией XOR. Отметим, что повторное объединение восстанавливает первоначальное изображение на канве. Это используется в играх, при движении какого-то объекта по фону |

| | |
|---|---|
| cmSrcPaint | Объединяет изображение на канве и источнике операцией OR |
| CmWhiteness | Заполняет область рисования белым цветом |

Канва не является компонентом, но во многих компонентах является свойством. С помощью свойства Pixels все пиксели канвы представляются в виде двумерного массива точек. Изменяя цвет пикселей, можно прорисовывать изображение по отдельным точкам.

| Методы класса Таблица 3 | |
|---|---|
| Procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer); | Чертит дугу эллипса в охватывающем прямоугольнике (X1, Y1)- (X2, У2). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (X3, Y3), а конец - на пересечении с лучом из центра в точку (X4, У4). Дуга чертится против часовой стрелки (рис.3.1) |
| Procedure Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer) ; | Чертит сегмент эллипса в охватывающем прямоугольнике (XI, VI) - (X2, У2). Начало дуги сегмента лежит на пересечении эллипса и луча, проведенного из его центра в точку (X3, Y3), а конец - на пересечении с лучом из центра в точку (X4, У4), Дуга сегмента чертится против часовой стрелки, а начальная и конечная точки дуги соединяются прямой (рис. 3.2) |
| Procedure CopyRect(Dest: Trect; Canvas: Tcanvas; Source: Trect); | Копирует изображение Source канвы Canvas в участок Dest текущей канвы. При этом свойство CopyMode определяет различные эффекты копирования |
| Procedure Draw(X, Y: Integer; Graphic: TGraphic); | Осуществляет вывод на канву графического объекта Graphic так, чтобы левый верхний угол объекта расположился в точке (X, Y) |
| Procedure E11ipse(X1, Y1, X2, Y2: Integer); | Чертит эллипс в охватывающем прямоугольнике (X1, Y1) -(X2, Y2). Заполняет внутреннее пространство эллипса текущей кистью |
| Procedure FillRect(const Rect: Trect) ; | Заполняет текущей кистью прямоугольную область Rect, включая ее левую и верхнюю границы, но не затрагивая правую и нижнюю границы |
| Procedure FloodFill(X, Y: Integer; Co1or: TCo1or; FillStyle: TFillStyle) ; | Производит заливку канвы текущей кистью. Заливка начинается с точки (X, У) и распространяется во все стороны от нее. Если FillStyle=fsSurface, заливка распространяется на все соседние точки с цветом Co1or. Если FillStyle=fsBorder, наоборот, заливка прекращается на точках с этим цветом |

Продолжение табл. 3

| | |
|---|---|
| Procedure FrameRect(const Rect:TRect) ; | Очерчивает границы прямоугольника Rect текущей кистью толщиной в 1 пиксель без заполнения внутренней части прямоугольника |

| | |
|---|---|
| Procedure LineTo (X, У: Integer); | Чертит линию от текущего положения пера до точки (X. У) |
| Procedure MoveTo(X, У: Integer) ; | Перемещает карандаш в положение (X, У) без вычерчивания линий |
| Procedure Pie(X1, У1, X2, У2, X3, У3, X4, У4: Integer); | Рисует сектор эллипса в охватывающем прямоугольнике (X1, У1) - (X2, y2). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (X3, У3), а конец - на пересечении с лучом из центра в точку (X4, У4). Дуга чертится против часовой стрелки. Начало и конец дуги соединяются прямыми с ее центром (см. рис. 3.3,в) |
| Procedure Po1ygon (Points: array of TPoint) ; | Вычерчивает карандашом многоугольник по точкам, заданным в массиве Points. Конечная точка соединяется с начальной и многоугольник заполняется кистью. Для вычерчивания без заполнения используйте метод Po1yline. |
| Procedure Po1y1ine (Points: array of TPoint) ; | Вычерчивает карандашом ломаную прямую по точкам, заданным в массиве Poimts. |
| Procedure Rectangle (X1, У1, X2, У2: Integer); | Вычерчивает и заполняет прямоугольник (X1, У1) - (X2, У2). Для вычерчивания без заполнения используйте FrameRect или Po1yLine. |
| Procedure ReFresh; | Устанавливает в канве умалчиваемые шрифт, карандаш и кисть. |
| Procedure RoundRect (X1, У1, X2, У2, X3, У3: Integer) ; | Вычерчивает и заполняет прямоугольник (XI, У1) - (X2, У2) со скругленными углами. Прямоугольник (X1, Y1) - (X3, У3) определяет дугу эллипса для округления углов (рис.13.3,г) |
| Procedure StretchDraw (const Rect: TRect; Graphic: TGraphic ) ; | Вычерчивает и при необходимости масштабирует графический объект Graphic так, чтобы он полностью занял прямоугольник Rect |
| Procedure TextOut (X, У: Integer; const Text: String) ; | Выводит текстовую строку Text так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке (X, У) |
| Procedure TextRect (Rect: TRect; X, У: Integer; const Text: String) ; | Выводит строку Text так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке (X, У). Если при этом какая-либо часть надписи выходит из границ прямоугольника Rect, она отсекается и не будет видна |

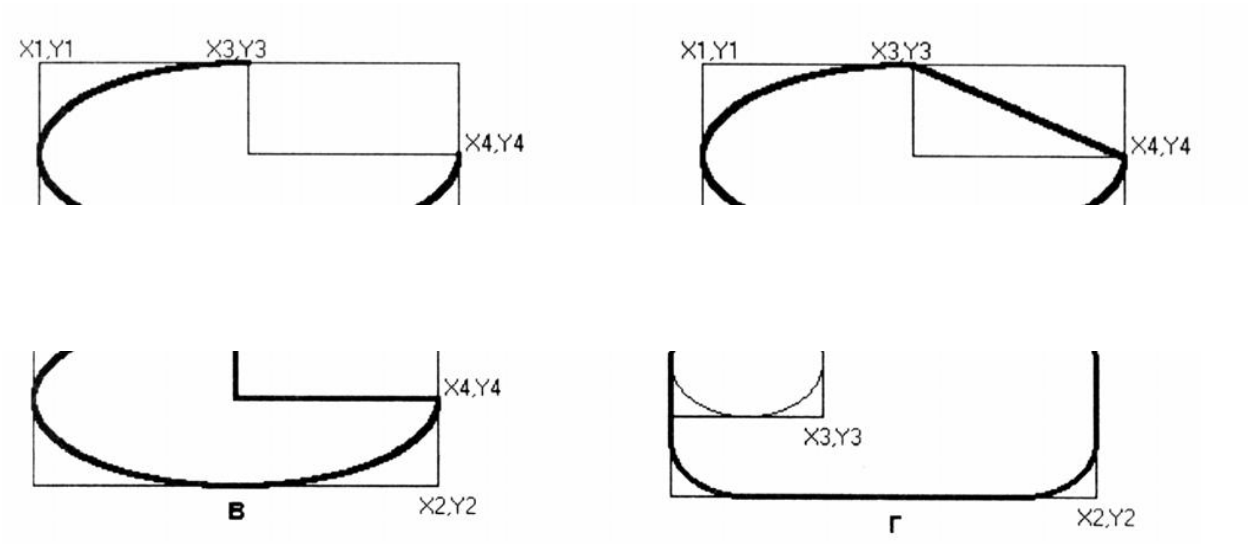


Рис. 3.1. Параметры обращения: а) к методу Arc; б) к методу Chord;
в) к методу Pie; г) к методу RoundRect

*3.2. Классы TGraphic и TPicture*

Важное место в графическом инструментарии Delphi занимают классы TGraphic и Tpicture.

Tgraphic - это абстрактный класс, инкапсулирующий общие свойства и методы трех своих потомков: пиктограммы (TIcon), метафайла (TmetaFile) и растрового изображения (TBitMap). Общей особенностью потомков TGraphic является то, что обычно они сохраняются в файлах определенного формата. Пиктограммы представляют собой небольшие растровые изображения, снабженные специальными средствами, регулирующими их прозрачность. Для файлов пиктограмм обычно используется расширение ICO. Метафайл - это изображение, построенное на графическом устройстве с помощью специальных команд, которые сохраняются в файле с расширением WMF или EMF. Растровые изображения - это произвольные графические изображения в файлах со стандартным расширением BMP.

Окончание табл. 4

| | |
|---|---|
| Property Height: Tnteger; | Содержит высоту изображения в пикселях |
| Property Modefied: Boo1ean; | Содержит *Tгue,* если графический объект изменялся |
| Property Palette: HPALETTE; | Содержит цветовую палитру графического объекта |
| Property PaletteModified: Boolean; | Содержит *True*, если менялась цветовая палитра графического объекта |
| Property Transparent: Boolean; | Содержит *True*,если объект прозрачен для фона, на котором он изображен |
| Property Width: Integer; | Содержит ширину изображения в пикселях |

Окончание табл. 6

| | |
|---|---|
| Property Graphic: TGraphic; | Содержит графический объект |
| Property Height: Integer; | Содержит высоту изображения в пикселях |
| Property Icon: TIcon; | Интерпретирует графический объект как пиктограмму |
| Property Metafile: TMetafile; | Интерпретирует графический объект как метафайл |
| Property Width: Integer; | Содержит ширину изображения в пикселях |

Окончание табл. 7

| | |
|---|---|
| Procedure SaveToFile(const FileName: String) ; | Сохраняет изображение в файле *FileName* |
| class Function SupportsClipboard-Format(AFormat: Word): Boolean; | Возвращает *True,* если формат *AFormat* зарегистрирован в буфере межпрограммного обмена *Clipbord* |
| class Procedure UnregisterGraphic-Class(AClass: TGraphicClass); | Делает недоступными любые графические объекты класса *AClass* |

**Задания**

1. Разработать программу, реализующую игру «Бега лошадей по кругу ипподрома». Предусмотреть возможность устанавливать ставки на лошадей и расчета выигрыша. Скорость движения лошадей должна задаваться случайным образом функцией Random.
2. Разработать программу, реализующую игру «Бега лошадей по прямой». Предусмотреть возможность устанавливать ставки на лошадей и расчета выигрыша. Скорость движения лошадей должна задаваться случайным образом функцией Random.
3. Разработать программу игры в крестики – нолики. В основу положить компонент DrawGrid.
4. Разработать программу игры «Минер» по подобию такой же игры в системе Windows. Начальная расстановка мин должна выполняться случайным образом. В основу положить компонент DrawGrid.
5. Разработать программу игры «Стрельба из подводной лодки по кораблю», используя вид из перископа. На заднем плане должен периодически проплывать кораблик с постоянной поперечной скоростью. С помощью клавиш влево-вправо следует менять вид в перископе. Клавиша «Ввод» должна запускать торпеду. В перископе должна отображаться траектория движения торпеды с уменьшением скорости движения при приближении к кораблю. Попадание должно сопровождаться видимым взрывом и исчезновением корабля.
6. Разработать программы игры «Бомбометание с самолета по наземной цели». С летящего с постоянной скоростью самолета клавишей «Ввод» производить бомбометание. Траектория движения бомбы должна соответствовать физическим законам падения тел на землю. Попадание в цель должно сопровождаться видимым взрывом и исчезновением цели. Самолет должен периодически вылетать из-за края канвы компонента рисования.
7. Разработать программу игры «Морской бой». Программа должна случайным образом на сетке 10х10 расставлять корабли: один четырехклеточный, два трехклеточных, три двухклеточных и четыре одноклеточных. Они не могут изгибаться и соприкасаться друг с другом. Игрок выбирает определенный квадрат и как бы стреляет в него. Программа должна сообщать, попал ли игрок в корабль или нет. Она также должна отображать все старые выстрелы и показывать ячейки, куда уже не имеет смысла стрелять. Аналогично строится и вторая таблица, где игрок располагает свои корабли, по которым уже случайным образом стреляет программа. Выигрывает тот, кто быстрей потопит корабли неприятеля. Предусмотреть в конце игры показ расположения кораблей программы. Для таблиц использовать компоненты TdrawGrid.
8. Разработать программы игры «Стрельба из пушки». Пушка должна стрелять через гору по какой-то цели. Траектория полета снаряда должна подчиняться законам физики. Игрок может управлять углом подъема ствола относительно горизонта и начальной скоростью снаряда в дискретных величинах (определяется типом снаряда). При попадании должен происходить видимый взрыв и исчезновение цели.
9. Разработать программу показа в форме текущего времени в виде обычных стрелочных часов со стрелками часов, минут и секунд.

10. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox произвольные кривые с помощью мыши. Предусмотреть возможность:
    а) изменения толщины кривых,
    б) изменение цвета кривых,
    в) сохранение рисунка в графическом файле.
11. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox ломанные линии с помощью нажатия на клавиши мыши. Предусмотреть возможность:
    а) изменения толщины линий,
    б) изменение цвета линий,
    в) сохранение рисунка в графическом файле.
12. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox с помощью мыши прямоугольники. Предусмотреть возможность:
    а) изменения толщины линий,
    б) изменение цвета линий,
    в) заливку областей текущей кистью,
    г) изменение цвета кисти,
    д) сохранение рисунка в графическом файле.
13. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox с помощью мыши эллипсы. Предусмотреть возможность:
    а) изменения толщины линий,
    б) изменение цвета линий,
    в) заливку областей текущей кистью,
    г) изменение цвета кисти,
    д) сохранение рисунка в графическом файле.
14. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox любой текст в указанном мышкой месте. Предусмотреть возможность:
    а) изменения типа, размера и цвета шрифта,
    б) сохранение рисунка в графическом файле.
15. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен помещать в канву компонента TpaintBox из графического файла произвольный рисунок и обеспечивать возможность:
    а) стирания произвольной области рисунка,
    б) изменение размеров стирки,

в) сохранение рисунка в графическом файле.

**Лабораторная работа №3**
Интегрированная среда разработки языка программирования Delphi.

## *1.1. ЦЕЛЬ РАБОТЫ*

Изучение основных элементов инструментальной среды разработки Delphi/C++ Builder. Ознакомление с понятием проекта в Delphi/C++ Builder. Получение сведений о назначении файлов проекта. Освоение приемов добавления и удаления визуальных компонентов на форму; редактирования свойств визуальных компонент с помощью инспектора объектов; добавления обработчиков событий. Получение навыков разработки проектов: компиляция проекта; запуск программы из интегрированной среды; пошаговая отладка с инспектированием значений переменных; создание, добавление и удаление модулей и форм; настройка опций проекта, настройка опций среды программирования.

## *1.2. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ*

Borland Delphi, Borland C++ Builder. Внешний вид и набор основных элементов среды программирования Borland Delphi и Borland C++ Builder идентичны. Библиотеки визуальных компонентов этих инструментальных сред также не имеют отличий. Поэтому студент имеет право выбрать по своему усмотрению в качестве среды разработки либо Borland Delphi, либо C++ Builder.

## *1.3. СРЕДА ПРОГРАММИРОВАНИЯ BORLAND DELPHI*

### *1.3.1.* Проект Delphi

Любой проект имеет, по крайней мере, шесть файлов, связанных с ним. Три из них относятся к управлению проектом из среды и напрямую программистом не меняются. Вот эти файлы:

- Главный файл проекта, изначально называется PROJECT1.DPR.
- Первый модуль программы /unit/, который автоматически появляется в начале работы. Файл называется UNIT1.PAS по умолчанию, но его можно назвать любым другим именем, вроде MAIN.PAS.
- Файл главной формы, который по умолчанию называется UNIT1.DFM, используется для сохранения информации о внешнем виде главной формы.
- Файл PROJECT1.RES содержит иконку для проекта, создается автоматически.
- Файл, который называется PROJECT1.OPT по умолчанию, является текстовым файлом для сохранения установок, связанных с данным проектом. Например, установленные Вами директивы компилятора сохраняются здесь.
- Файл PROJECT1.DSK содержит информацию о состоянии рабочего пространства.

Если сохранить проект под другим именем, то изменят название и файлы с расширением RES, OPT и DSK.

После компиляции программы получаются файлы с расширениями:

- DCU - скомпилированные модули
- EXE - исполняемый файл
- DSM - служебный файл для запуска программы в среде, очень большой, рекомендуется стирать его при окончании работы.

- ~PA, ~DP - backup файлы Редактора.

### *1.3.2.* Структура среды программирования

Внешний вид среды программирования Delphi отличается от многих других приложений Windows. Среда Delphi состоит из нескольких отдельно расположенных окон. Перед началом работы приложения типа Delphi лучше минимизировать другие приложения, чтобы их окна не загромождали рабочее пространство.

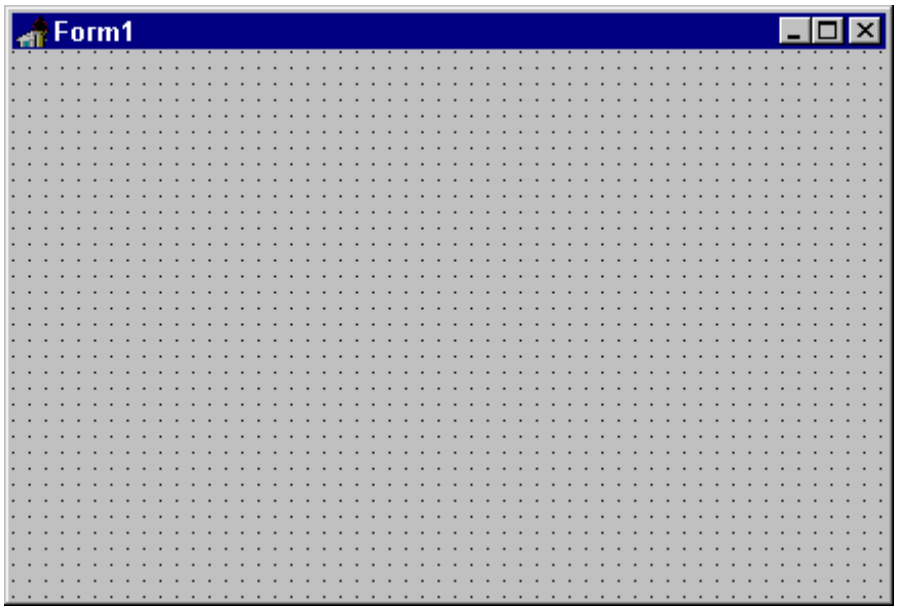


Рисунок 1.1 – Редактор форм

Главными составными частями среды программирования Delphi являются:

1. Редактор Форм (Form Designer)
2. Окно Редактора Исходного Текста (Editor Window)
3. Палитра Компонент (Component Palette)
4. Инспектор Объектов (Object Inspector)
5. Справочник (On-line help)

При разработке приложения в Delphi наиболее активно используются Редактор Форм (показан на рисунке 1.1) и Окно Редактора Исходного Текста (показан на рисунке 1.2).

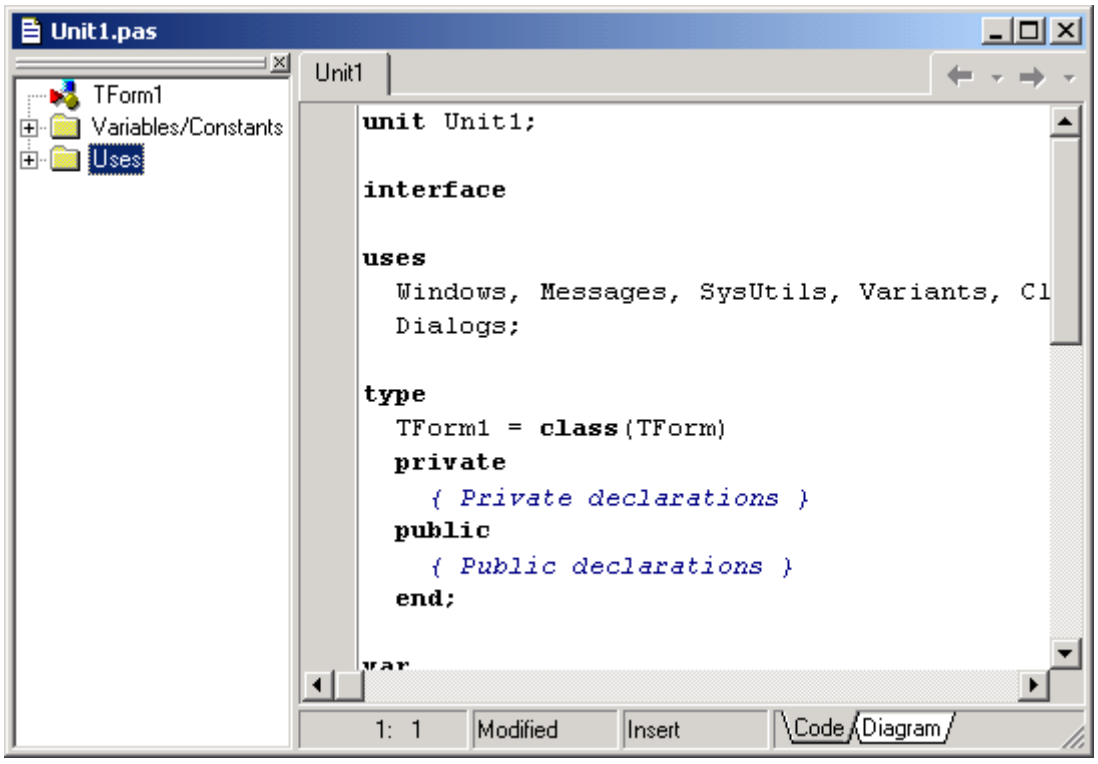


Рисунок 1.2 – Окно редактора текста

Редактор Форм - это то место, где создается визуальный интерфейс программы в Delphi. Редактор интуитивно понятен и прост в использовании. Первоначально он состоит из одного пустого окна, которое заполняется объектами, выбранными программистом на Палитре Компонент.

В окне Редактора программист создает логику управления программой.

Палитра Компонент (рисунок 1.3) позволяет выбрать нужные компоненты для размещения их на Дизайнере Форм. Для использования Палитры Компонент достаточно первый раз щелкнуть мышкой на один из объектов и потом второй раз - на Дизайнере Форм. Выбранный объект появится на проектируемой форме. Им можно манипулировать с помощью мыши: двигать с места на место, использовать границу, прорисованную вокруг объекта для изменения его размеров. Однако, невидимые во время выполнения программы компоненты (TDataBase) не меняют своей формы.


Рисунок 1.3 – Палитра Компонент

Палитра Компонент использует постраничную группировку объектов. Внизу Палитры находится набор закладок - Standard, Additional и т.д.

Слева от Дизайнера Форм располагается Инспектор Объектов (рисунок 1.4). Информация в Инспекторе Объектов меняется в зависимости от объекта, выбранного на форме. Каждый компонент является настоящим объектом. Вид и поведение компонента можно менять с помощью Инспектора Объектов.

Инспектор Объектов состоит из двух страниц. Первая страница - это список свойств (предназначена для определения внешнего вида объекта), вторая - список событий (предназначена для определения поведения).

Страница событий связана с Редактором текста; если дважды щелкнуть мышкой на правую сторону какого-нибудь имени события, то соответствующий данному событию шаблон автоматически запишется в Редактор, сам Редактор немедленно получит фокус, что дает возможность добавить код обработчика данного события.

Последняя важная часть среды Delphi - Справочник (on-line help). Для доступа к этому инструменту нужно выбрать в системном меню пункт Help и затем Delphi Help. На экране появится

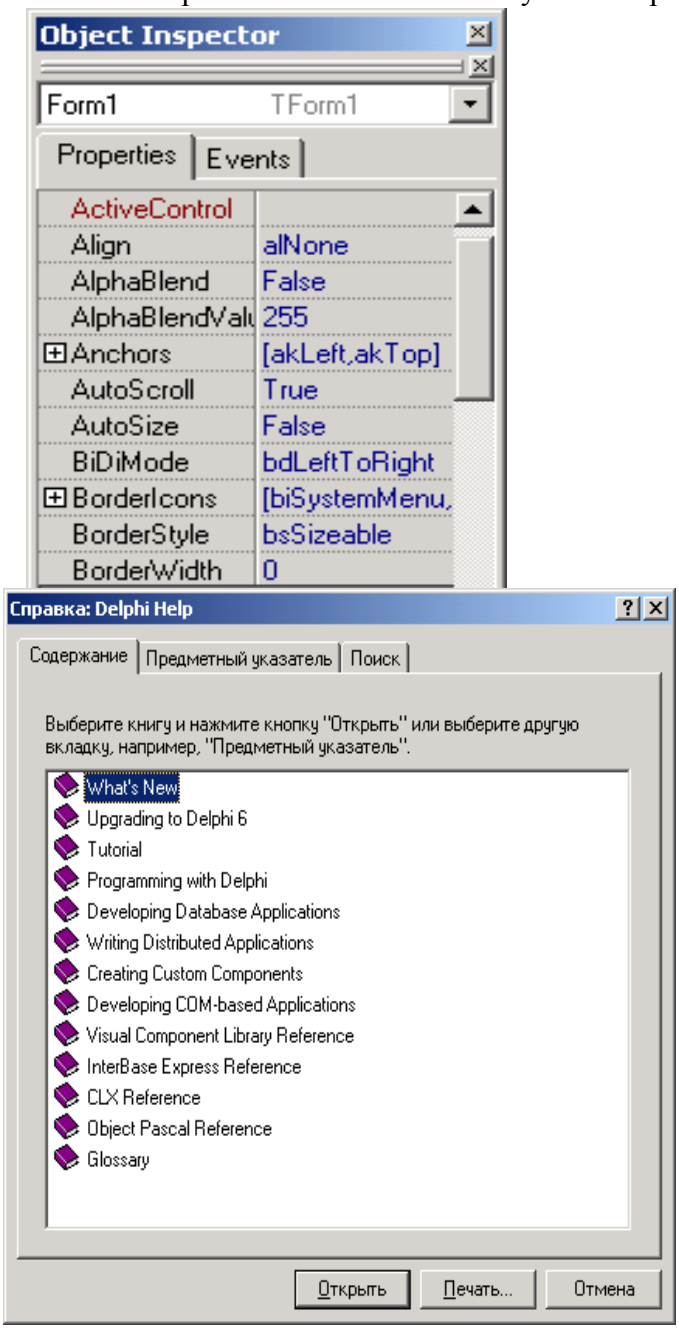


Рисунок 1.5 – Справочник Delphi

Справочник, показанный на рисунке 1.5.
Справочник является контекстно-зависимым; при нажатии клавиши F1, выдается подсказка, соответствующая текущей ситуации.

В Delphi имеются инструменты, которые можно воспринимать как вспомогательные для среды программирования:

- Меню (Menu System)
- Панель с кнопками для быстрого доступа (SpeedBar)
- Редактор картинок (Image Editor)

Меню можно использовать для выполнения широкого круга задач; чаще всего, для наиболее общих задач вроде открытия и закрытия файлов, управления отладчиком или настройкой среды программирования.

*Пункт меню «File»* включает шесть секций:

- Первая секция дает возможность управления проектом в целом.
- Вторая секция дает контроль над формами, модулями и компонентами проекта.
- Третья позволяет добавлять и удалять файлы из проекта.
- Четвертая управляет печатью.
- Пятая секция - выход из Delphi
- Шестая секция предоставляет список ранее редактировавшихся проектов.

Большинство операций из пункта меню «File» можно выполнить с помощью Менеджера Проекта (Project Manager), который можно вызвать из пункта меню «View».

Менеджер Проектов помогает управлять проектом. Менеджер Проектов (рисунок 1.6) разделен на две части. Верхняя - панель с управляющими кнопками. Нижняя - список модулей, входящих в проект.

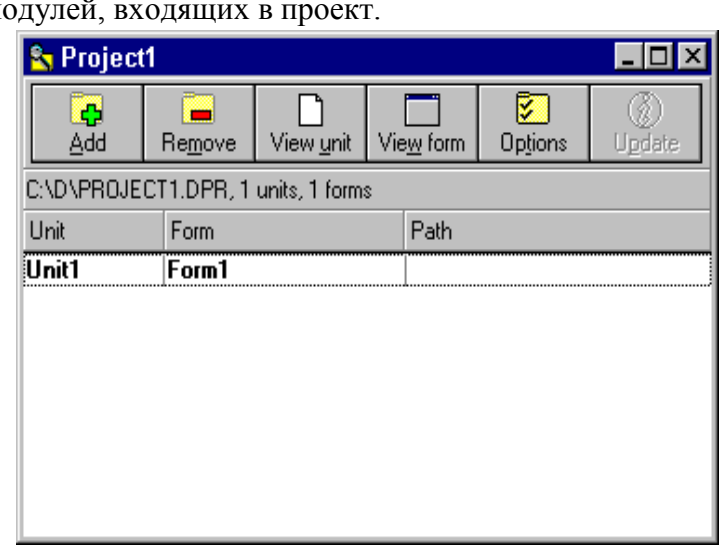


Рисунок 1.6 – Менеджер проектов

Кнопки с плюсом и минусом используются для добавления и удаления файлов в проекте. Эти изменения влияют на файлы с исходным текстом, то есть, если добавить в проект модуль, то ссылка на него появится в файле с расширением DPR.

Краткое описание других кнопок:

- Кнопка View unit - просмотр текста модуля, на котором стоит курсор.
- Кнопка View form - просмотр формы, если есть таковая для данного модуля
- Кнопка Options - вызов диалога настройки проекта.

Кнопка Update - сохранение изменений на диске.
Некоторые операции доступны и через SpeedBar. Данная стратегия типична для Delphi: она предоставляет несколько путей для решения одной и той же задачи.

Первые два пункта второй секции позволяют создать новую форму или новый модуль.

Пункт «New Component» вызывает диалог для построения заготовки нового визуального компонента. В результате создается модуль, который можно скомпилировать и включить в Палитру Компонент.

Пункт «Open File» открывает при необходимости любой модуль или просто текстовый файл. Если модуль описывает форму, то эта форма тоже появится на экране.

При создании нового модуля Delphi дает ему имя по - умолчанию. Можно изменить это имя с помощью пункта «Save File As».

«Save File» сохраняет только редактируемый файл, но не весь проект.

«Close File» удаляет файл из окна Редактора.

*Пункт меню «Edit»* содержит команды «Undo» и «Redo», которые могут быть очень полезны при работе в редакторе для устранения последствий при неправильных действиях, например, если случайно удален нужный фрагмент текста. Команды «Cut», «Copy», «Paste» и «Delete» - как во всех остальных приложениях Windows, но их можно применять не только к тексту, но и к визуальным компонентам.

*Пункт меню «Search»* содержит команду «Find Error» (поиск ошибки), которая поможет отследить ошибку периода выполнения программы.

Составляющие *пункта меню «View»*:

- Project Manager –менеджер проекта.
- Project Source - загружает главный файл проекта (DPR) в Редактор
- Установка, показывать или нет Object Inspector на экране.
- Установка, показывать или нет Alignment Palette. То же самое доступно из пункт меню Edit | Align.
- Browser - вызов средства для просмотра иерархии объектов программы, поиска идентификатора в исходных текстах и т.п.
- Watch, Breakpoint и Call Stack - связаны с процедурой отладки.
- Component List - список компонент, альтернатива Палитре Компонент. Используется для поиска компонента по имени или при отсутствии мыши.
- Window List - список окон, открытых в среде Delphi.
- Toggle Form/Unit, Units, Forms - переключение между формой и соответствующим модулем, выбор модуля или формы из списка.
- New Edit Window - открывает дополнительное окно Редактора. Полезно, если нужно, например, просмотреть две разных версии одного файла.
- SpeedBar и Component Palette - установки, нужно ли их отображать.

*Пункт меню «Compile»* позволяет скомпилировать (compile) или перестроить (build) проект. Если выбрать Compile или Run, то Delphi перекомпилирует только те модули, которые изменились со времени последней компиляции. Build all, с другой стороны, перекомпилирует все модули, исходные тексты которых доступны. Команда Syntax Check только проверяет правильность кода программы, но не обновляет DCU файлы.

В самом низу - пункт Information, который выдает информацию о программе: размеры сегментов кода, данных и стека, размер локальной динамической памяти и количество скомпилированных строк.

*Пункт меню «Run»* можно использовать для компиляции и запуска программы и для указания параметров командной строки для передачи в программу. Здесь же имеются опции для режима отладки.

*Пункт меню «Options»* наиболее сложная часть системного меню. Это центр управления, из которого можно изменять установки для проекта и для всей рабочей среды Delphi:

- Project - выбор установок, которые напрямую влияют на текущий проект.
- Environment - конфигурация самой среды программирования (IDE).
- Tools - позволяет добавить или удалить вызов внешних программ в пункт главного меню «Tools».
- Gallery - позволяет определить специфические установки для Эксперта Форм и Эксперта Проектов и их «заготовок». Эксперты и «заготовки» предоставляют путь для ускорения конструирования интерфейса программы.

Последние три пункта позволяют сконфигурировать Палитру Компонент.

Диалог из пункта Options | Project включает пять страниц:

На странице Forms (рисунок 1.7) можно выбрать главную форму проекта. Изменения, которые Вы сделаете, отобразятся в соответствующем файле DPR.

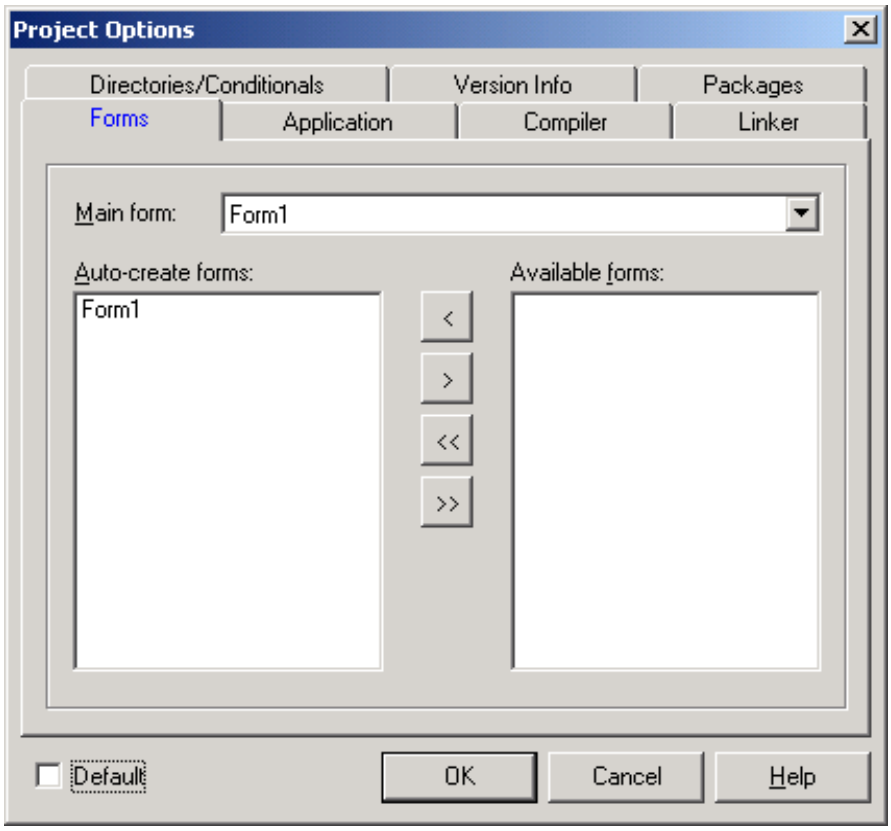


Рисунок 1.7 – Страница Forms

Можно также использовать эту страницу для определения, будет ли данная форма создаваться автоматически при старте программы. Если форма создается не автоматически, а по ходу выполнения программы, то для этого нужно использовать процедуру Create.

На *странице Applications* можно задать заголовок (Title), файл помощи (Help file) и пиктограмму (Icon) для проекта.

*Страница Linker* показана на рисунке 1.8.

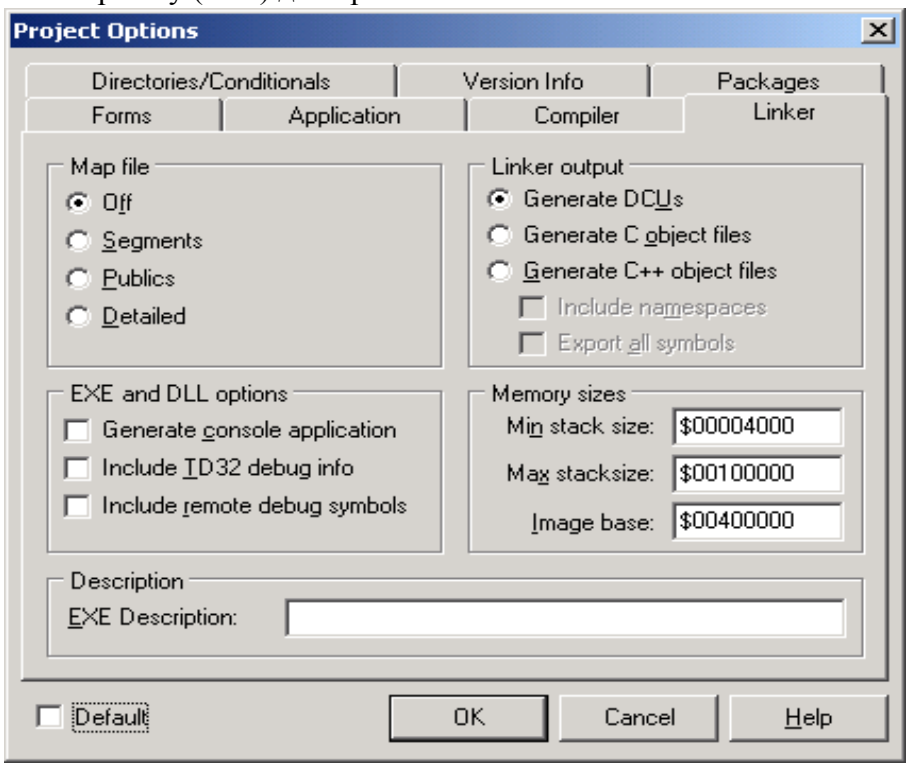


Рисунок 1.8 – Страница Linker

*Страница Directories/Conditionals* дает возможность расширить число директорий, в которых компилятор и линковщик ищут DCU файлы.

В файле DELPHI.INI содержится еще один список дире
кторий. В OPT файле - список директорий для конкретного проекта, а в файле DELPHI.INI - список относится к любому проекту.

- Output directory - выходная директория, куда складываются EXE и DCU файлы, получающиеся при компиляции.
- Search path - список директорий для поиска DCU файлов при линковке. Директории перечисляются через точку с запятой;
- Conditional defines - для опытного.

*Пункт меню «Environment Options ...»* предоставляет большой набор страниц и управляющих элементов, которые определяют внешний вид и работу IDE. Delphi позволяет сделать следующие важные настройки:

1. Определить, что из проекта будет сохраняться автоматически.
2. Можно менять цвета IDE.
3. Можно менять подсветку синтаксиса в Редакторе.
4. Можно изменить состав Палитры Компонент.
5. Указать “горячие клавиши” IDE.

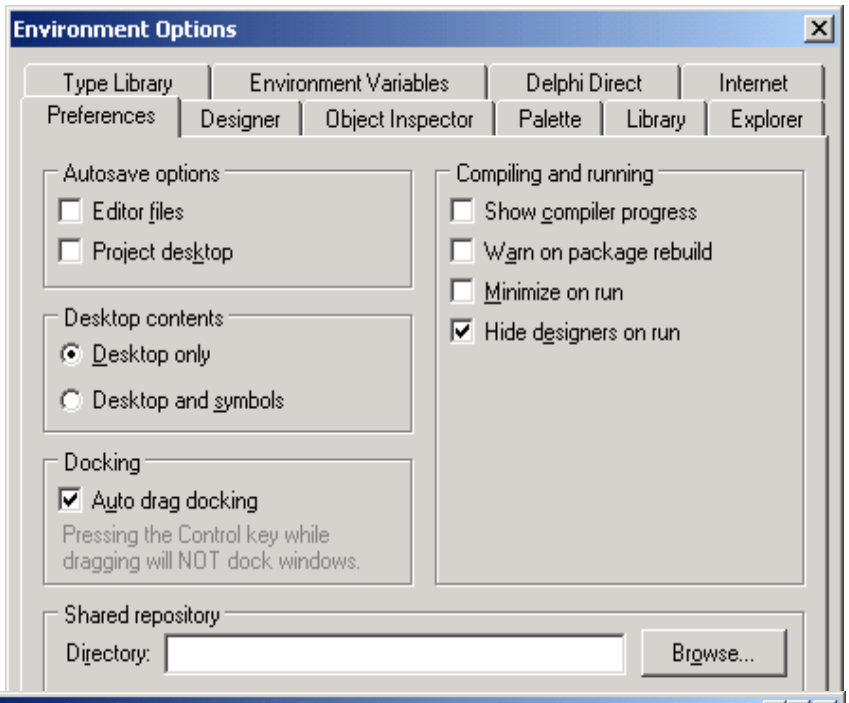



Рисунок 1.10 - Редактор Картинок

Первая страница пункта меню «Environment Options» показана на рисунке 1.9.

В группе «Desktop contents» определяется, что будет сохраняться при выходе из Delphi. Если выбрать Desktop Only - это сохранит информацию о директориях и открытых окнах, если выбрать Desktop And Symbols - это сохранит то же самое плюс информацию для браузера.

В группе «Autosave options» указывается, что нужно сохранять при запуске программы. Если позиция Editor Files выбрана, то сохраняются все модифицированные файлы из Редактора при выполнении команд Run|Run, Run|Trace Into, Run|Step Over, Run|Run To Cursor или при выходе из Delphi. Если позиция Desktop выбрана - сохраняется рабочая среда при закрытии проекта или при выходе из Delphi.

Редактор Картинок, показанный на рисунке

1.10, работает аналогично программе Paintbrush из Windows. Получить доступ к этому модулю можно, выбрав пункт меню Tools | Image Editor.

### *1.3.3.* Инструментальные средства

В дополнение к инструментам, рассмотренным выше, существуют пять средств, поставляемых вместе с Delphi. Эти инструментальные средства:

- Встроенный отладчик
- Внешний отладчик (поставляется отдельно)
- Компилятор командной строки
- WinSight

Данные инструменты собраны в отдельную категорию не потому, что они менее важны, чем другие, но потому, что они играют достаточно абстрактную техническую роль в программировании.

Отладчик позволяет пройти пошагово по исходному тексту программы, выполняя по одной строке за раз, и открыть просмотровое окно (Watch), в котором будут отражаться текущие значения переменных программы.

Встроенный отладчик, который наиболее важен из пяти вышеперечисленных инструментов, работает в том же окне, что и Редактор. Внешний отладчик делает все, что делает встроенный и выполняет дополнительные функции. Он более быстр и мощен, чем встроенный. Однако он не так удобен в использовании, главным образом из-за необходимости покидать среду Delphi.

Внешний компилятор (DCC.EXE) полезен, в основном, если необходимо скомпилировать приложение перед отладкой его во внешнем отладчике. Возможно создать и откомпилировать программу на Delphi, используя только DCC.EXE и еще одну программу CONVERT.EXE, которая поможет создать формы. Однако, данный подход неудобен для большинства программистов.

WinSight (WS.EXE) интересен преимущественно для опытных программистов в Windows. Этот инструмент используются для узких технических целей. Основная его функция – позволяет наблюдать за системой сообщений Windows. Хотя Delphi делает много для того, чтобы спрятать сложные детали данной системы сообщений от неопытных пользователей, тем не менее Windows является операционной системой, управляемой событиями. Почти все главные и второстепенные события в среде Windows принимают форму сообщений, которые рассылаются с большой интенсивностью между различными окнами на экране. Delphi дает полный доступ к сообщениям Windows и позволяет отвечать на них, как только будет нужно. В результате, опытным пользователям WinSight становится просто необходим.

### *1.3.4.* Стандартные компоненты

На первой странице Палитры Компонент размещены 14 объектов (рисунок 1.11) наиболее важных для использования (кнопка, списки, окно ввода и т.д.). Набор и порядок компонент на каждой странице являются конфигурируемыми. Можно добавить к имеющимся компонентам новые, изменить их количество и порядок.


Рисунок 1.11 – Компоненты страницы Standard

Стандартные компоненты Delphi:

- TMainMenu позволяет поместить главное меню в программу. При помещении TMainMenu на форму это выглядит не как меню, а как обычная иконка. Иконки данного типа называют «невизуальными компонентами», поскольку они невидимы во время выполнения программы. Создание меню включает три шага:
  1. помещение TMainMenu на форму,
  2. вызов Дизайнера Меню через свойство Items в Инспекторе Объектов,
  3. определение пунктов меню в Дизайнере Меню.
- TPopupMenu позволяет создавать всплывающие меню. Этот тип меню появляется по щелчку правой кнопки мыши.
- TLabel служит для отображения текста на экране. Для изменения шрифта и цвета метки необходимо дважды щелкнуть на свойство Font в Инспекторе Объектов. Изменить данные свойства возможно и во время выполнения программы.
- TEdit - стандартный управляющий элемент Windows для ввода. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы.
- TMemo - иная форма TEdit. Подразумевает работу с большими текстами. TMemo может переносить слова, сохранять в Clipboard фрагменты текста и восстанавливать их, и другие основные функции редактора. TMemo имеет ограничения на объем текста в 32Кб, это составляет 10-20 страниц.
- TButton позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы. Поместив TButton на форму, по двойному щелчку можно создать заготовку обработчика события нажатия кнопки. Далее нужно заполнить заготовку кодом (подчеркнуто то, что нужно написать вручную):

```
procedure TForm1.Button1Click(Sender: TObject);
begin MessageDlg('Are you there?',mtConfirmation,mbYesNoCancel,0); end;
```

- TCheckBox отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что что-то выбрано.
- TRadioButton позволяет выбрать только одну опцию из нескольких.
- TListBox нужен для показа прокручиваемого списка.
- TComboBox во многом напоминает ListBox, за исключением того, что позволяет водить информацию в маленьком поле ввода сверху ListBox.
- TScrollbar - полоса прокрутки**,** появляется автоматически в объектах редактирования при необходимости прокрутки текста для просмотра.
- TGroupBox используется для визуальных целей и для указания Windows, каков порядок перемещения по компонентам на форме (при нажатии клавиши TAB).
- TPanel - управляющий элемент, похожий на TGroupBox, используется в декоративных целях. Чтобы использовать TPanel, нужно просто поместить его на форму и затем положить другие компоненты на него. Теперь при перемещении TPanel будут передвигаться и эти компоненты. TPanel используется также для создания линейки инструментов и окна статуса.
- TScrollBox представляет место на форме, которое можно скроллировать в вертикальном и горизонтальном направлениях. Используется в случаях, когда понадобится прокручивать только часть формы.

## 1.4. *ПРАКТИЧЕСКОЕ ЗАДАНИЕ*

Создать новый проект. Потренируйтесь добавлять и удалять визуальных компонентов на форму. Поработайте со свойствами объектов с помощью инспектора объектов. Добавьте обработчики событий. Запустите программу из интегрированной среды. Выполните пошаговую отладку с инспектированием значений переменных. Потренируйтесь в создании, добавлении и удалении модулей и форм. Выполните настройку опций проекта, а также настройку опций среды программирования. Сохраните проект.

## 1.5. *МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ*

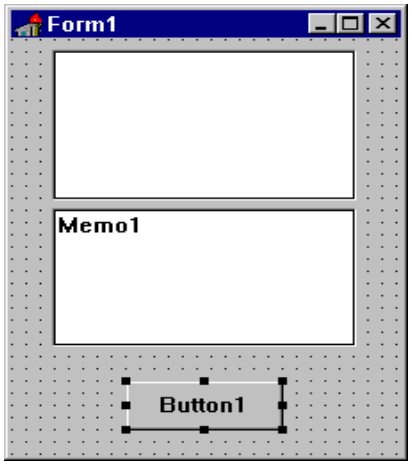


Рисунок 1.12 – Образец формы

Открыть новый проект можно, выбрав пункт меню File | New Project. Для работы можно создать форму согласно образцу (рисунок 1.12).

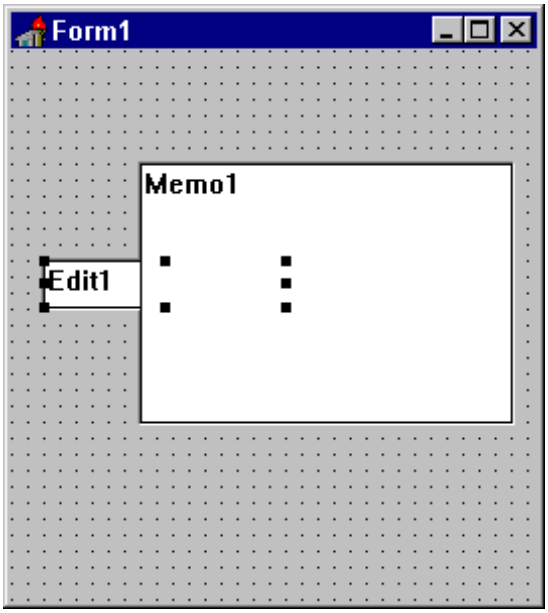


Рисунок 1.13 – Образец формы

Поместите на форму объект TMemo, а затем TEdit так, чтобы он наполовину перекрывал TMemo, как показано на рисунке 1.13. Теперь выберите пункт меню Edit | Send to Back, что приведет к перемещению TEdit вглубь формы, за TMemo. Это называется изменением порядка Z-объект компонент. Поместите кнопку TButton в нижнюю часть формы. Теперь растяните Инспектор Объектов так, чтобы свойства Name и Caption были видны одновременно на экране. Теперь измените имя кнопки на Terminate. Текст, который Вы видите на поверхности кнопки - это значение свойства Caption. Напишите обработчик нажатия на кнопку, который будет закрывать приложение.

Для работы со свойствами нажмите клавишу <Shift> и щелкните на TMemo и затем на TListBox. Теперь оба объекта имеют по краям маленькие квадратики, показывающие, что объекты выбраны. Выбрав два или более объектов одновременно, можно выполнить большое число операций над ними, например, передвигать по форме. Затем попробуйте выбрать пункт меню Edit | Size и установить оба поля

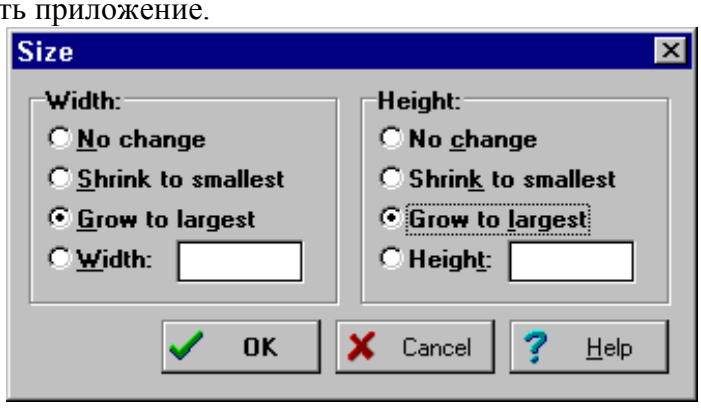


Рисунок 1.14 – Диалог Size

Ширину(Width) и Высоту(Height) в Grow to Largest**,** как показано на рисунке 1.14. Теперь оба объекта стали одинакового размера.

Затем выберите пункт меню Edit | Align и поставьте в выравнивании по горизонтали значение Center (рисунок 1.15).

Поскольку выбрано два компонента, то содержимое Инспектора Объектов изменится - он будет показывать только те поля, которые являются общими для объектов. Это означает то, что изменения в свойствах повлияют не на один, а на все выбранные объекты.

Рассмотрим изменение свойств объектов на примере свойства Color. Есть три способа изменить его значение в Инспекторе Объектов. Первый - просто напечатать имя цвета (clRed) или номер цвета. Второй путь - нажать на маленькую стрелку справа и выбрать цвет из списка. Третий путь - дважды щелкнуть на поле ввода свойства Color. При этом появится диалог выбора цвета.

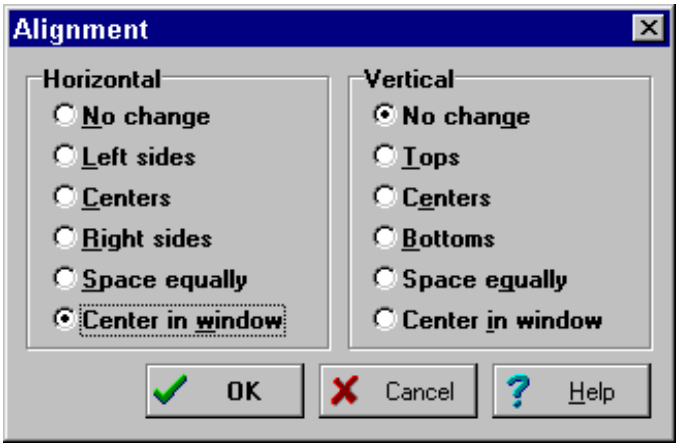


Рисунок 1.15 – Диалог Alignment

Свойство Font работает аналогично свойству Color.

Дважды щелкните на свойство Items объекта ListBox. Появится диалог, в котором Вы можете ввести строки для отображения в ListBox.

Напечатайте несколько слов, по одному на каждой строке, и нажмите кнопку OK. Текст отобразится в ListBox’е.

Сохранение программы:

Создать поддиректорию для программы. Лучше всего создать директорию, где будут храниться все Ваши программы и в ней создать поддиректорию для данной конкретной программы.

После создания поддиректории для хранения программы нужно выбрать пункт меню File | Save Project. Сохранить нужно будет два файла. Первый - модуль (unit), над которым Вы работали, второй - главный файл проекта, который "владеет" Вашей программой. Сохраните модуль под именем MAIN.PAS и проект под именем TIPS1.DPR.

Для разработки обработчика нажатия на кнопку перейдите на форму и дважды щелкните мышкой на объект TButton. Вы попадете в окно Редактора, в котором будет фрагмент кода:

```
procedure TForm1.TerminateClick(Sender: TObject);
begin
end;
```

Данный код был создан автоматически и будет выполняться всякий раз, когда во время работы программы пользователь нажмет кнопку Terminate. Определение класса в начале файла теперь включает ссылку на метод TerminateClick:

```
TForm1 = class(TForm)
    Edit1: TEdit;
    Memo1: TMemo;
    Terminate: TButton;
procedure TerminateClick(Sender: TObject);
    private
    { Private declarations }
    public
    { Public declarations }
```

```
end;
```

Для того, чтобы приложение закрывалось при нажатии на кнопку необходимо написать код:

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

**Лабораторная работа №4**
Этапы разработки проектов на языке Delphi.

Разработка приложений в Delphi означает работу с проектами. Иначе говоря, когда вы приступаете к разработке собственной программы в Delphi, первым делом создается проект - группа файлов, представляющих исходные данные (прежде всего, 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. Таким образом, все файлы проекта подразделяются на следующие типы:

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.

Основными составными частями проекта, помимо самого файла проекта (dpr), являются модули pas и dfm. При этом для каждого модуля окна (dfm) имеется собственный программный модуль (pas).

Чтобы лучше во всем этом разобраться, попробуем создать собственный проект в Delphi. Для этого достаточно запустить Delphi - в том случае, если вы не изменяли настроек, новый проект создастся автоматически. Но на всякий случай, мы все-таки создадим новый проект самостоятельно, для чего следует из меню File зайти в группу New и выбрать в ней пункт Application.

ВНИМАНИЕ
Чтобы постоянно не повторяться, в дальнейшем подобные последовательности действий мы будем обозначать следующим образом: File > New > Application.

В результате мы получим новый проект, полностью готовый к дальнейшему использованию. Более того, его уже на этом этапе можно выполнить! Для этого достаточно нажать на кнопку Run, находящуюся на панели инструментов отладки, или же выбрать пункт Run из одноименного меню (Run > Run), но лучше всего нажать на клавишу F9: для быстрой разработки приложений в среде Delphi знание хотя бы основных сочетаний "горячих клавиш" просто необходимо.

Итак, мы запустили приложение, правда, выглядеть оно будет довольно скучно: пустое окно с заголовком Form1 и стандартными кнопками управления окном (рис. 2.4). Но, по крайней мере, даже такое приложение обладает всей базовой функциональностью: его можно развернуть на весь экран, или наоборот, свернуть в панель задач, перемещать по экрану, изменять размеры, и, что самое главное - закрыть.

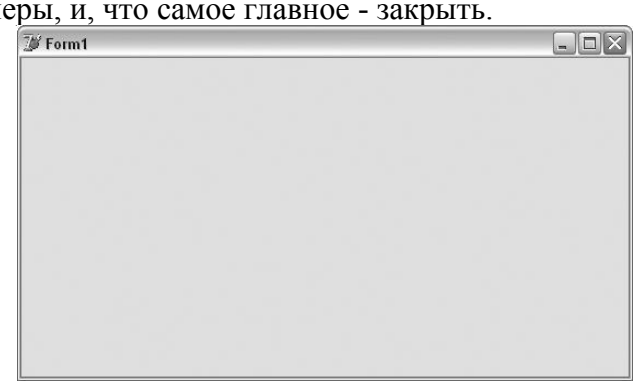


Рис. 2.4. Самое первое приложение в Дельфи

Теперь немного модернизируем свое приложение, заодно изучив еще одно важное окно среды Delphi - Object Inspector. Для этого вернитесь в рабочую среду Delphi, закрыв запущенное приложение, и щелкните по окну Form1, чтобы оно стало актив-ным. Это окно, как и любые другие окна, относящиеся непосредственно к разрабаты-ваемому приложению, называют формой. Теперь обратите внимание на окно инспек-тора объекта (рис. 2.5), по умолчанию оно расположено по левому краю экрана.

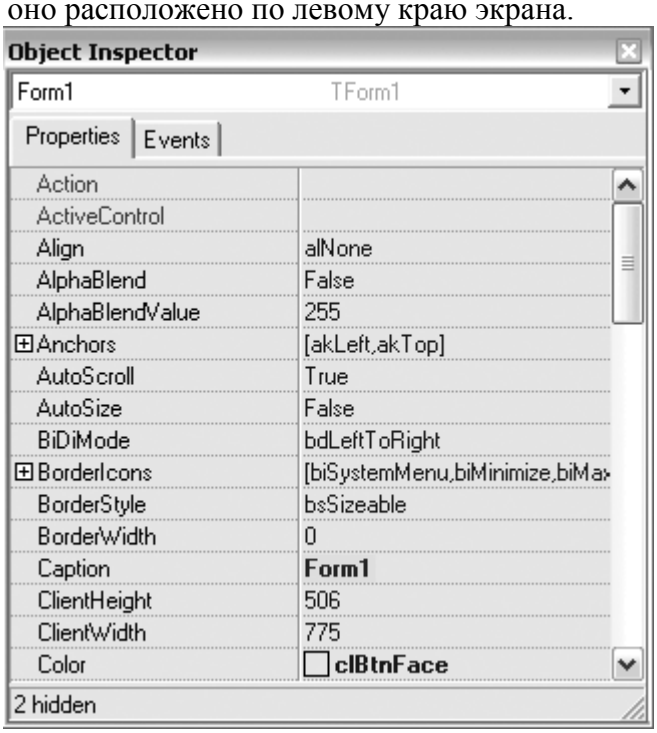


Рис. 2.5. Окно инспектора объектов

В его верхней части находится ниспадающий список, содержащий все элементы, интерфейса выбранной формы, в данном случае там будет только сама форма, отмеченная в виде имени объекта (Form1). Далее располагается список всех свойств объекта, доступных для изменения в процессе визуальной разработки.

ПРИМЕЧАНИЕ

Все свойства объектов в Delphi, доступные для изменения, делятся на run-time (времени выполнения) и на design-time (времени разработки). При этом первые можно изменять только во время работы программы, а вторые доступны уже во время визуального редактирования.

Попробуем заменить значение одного из свойств. Наиболее безопасным будет изменение такого свойства, как Caption - оно отвечает за текст, находящийся в заголовке окна. Для изменения значения этого свойства, щелкните по строке с ним и вместо "Form1" введите какой-либо собственный текст, например, "Мое первое приложение в Delphi". В данном случае вы сразу же увидите результат своей работы: заголовок окна формы изменится на новый.

Некоторые свойства меняются не путем непосредственного ввода значений, а путем выбора одного из заранее предусмотренных. В простейшем случае это может быть выбор ложь-истина (False или True), включающим или отключающим ту или иную опцию. Иногда списки бывают гораздо более объемными. Например, для выбора цвета предлагается множество цветов, включая системные. Например, свойство Color имеет в нашем случае значение clBtnFace, что означает цвет кнопки, установленный в настройках Windows. Мы можем изменить его на любой другой цвет, как системный (например, clCaptionText - цвет текста заголовка окна), так и на явный, например, clWhite (белый).

Можно заметить, что некоторые из свойств отмечены значком "+". Это означает, что такое свойство является составным, и если щелкнуть по значку, то раскроются строки, содержащие отдельные параметры. Например, можно раскрыть таким образом свойство

BorderIcons и поменять значение параметра biMinimize с True на False. Тем самым мы изменим значение свойства, отвечающего за системные элементы управления окна таким образом, что функция разворачивания окна на весь экран будет недоступной.

Есть и такие свойства, для редактирования которых открывается отдельное окно. Например, тот же цвет можно определить не выбирая его из списка предусмотрен-ных вариантов, а открыв стандартное для Windows окно выбора цвета. Для этого достаточно сделать двойной щелчок мышкой по области списка цвета. В других случаях (например, для выбора свойств шрифта - Font) окно настроек можно вызвать, нажав на имеющуюся напротив таких свойств кнопку с многоточием.

Ну а пока что сведем воедино все измененные нами свойства формы Form1:

Caption: Мое первое приложение в Delphi Color: clWhite BorderIcons: [biSystemMenu,biMinimize]

Последнее свойство - составное и такое его значение получается в результате установки таких его составляющих, как biSystemMenu и biMinimize в True, а biMaximize и biHelp - в False.

Часть из сделанных изменений - цвет окна и его заголовок - видна сразу же, в рабочей среде Delphi, т.е. на этапе разработки. А вот изменения в системных кнопках, хотя и могут быть сделаны на данном этапе, визуально себя не проявляют. Поэтому, чтобы увидеть сразу все произошедшие изменения, запустим приложение на выполнение, нажав клавишу F9, и вы увидите, что не только цвет и заголовок окна изменились, но и кнопка "развернуть" стала неактивной (рис. 2.6).

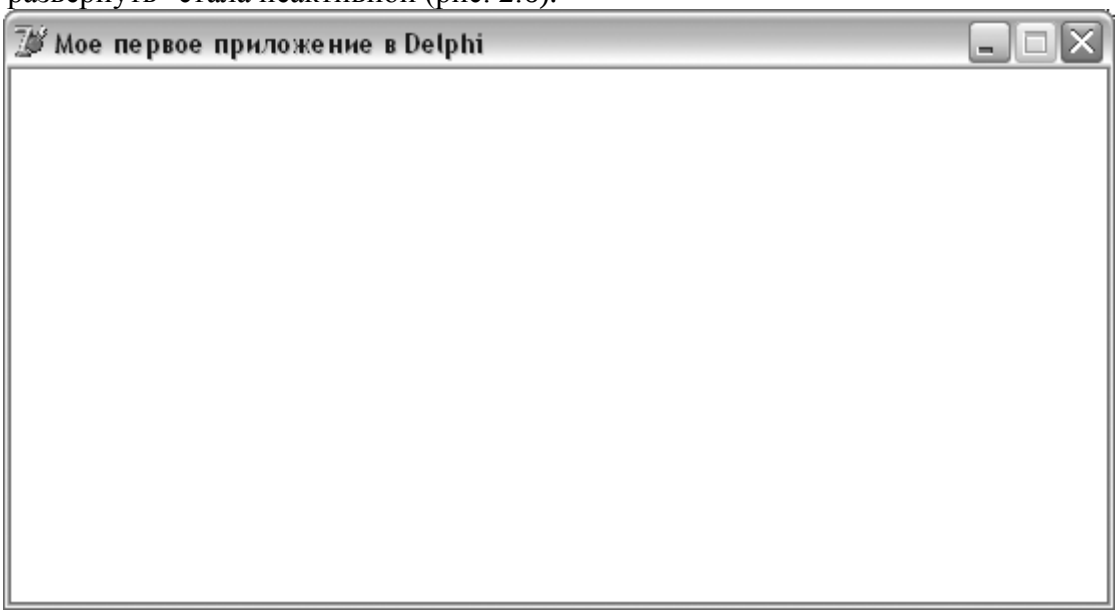


Рис. 2.6. Первое приложение после небольшой доработки

Таким образом, мы ознакомились с Object Inspector - одним из наиболее важных окон рабочей среды Delphi. Ну а чтобы завершить тему введения в проекты, попробуем сохранить наш проект на диске. Пусть это будет папка Project1, а сам проект мы назовем first. Для этого откройте диалог сохранения проекта (File Save project as) и выберите в нем нужную папку.

СОВЕТ

По умолчанию Delphi предлагает складывать все проекты в недра своей собственно директории в Program Files. Но было бы гораздо лучше, если бы вы создали папку в другом, легко доступном месте, и назвали ее понятным для себя названием. Например, это может быть папка Work на диске C:.

А теперь внимание! Если вы сохраняете проект в первый раз, то Delphi предложит вам сначала сохранить не файл проекта, а все несохраненные рабочие файлы. В данном случае это будет программный файл формы. По умолчанию Delphi предложит назвать его unit1.pas, но лучше сразу взять за правило давать осмысленные имена все рабочим

файлам. В частности, раз это окно - главное (и единственное) в нашем приложении, то назовем его файл main.pas. Таким образом будут сохранено сразу 2 файла - программный pas и файл формы dfm.

ВНИМАНИЕ

Имена любых файлов проекта должны состоять только из латинских букв и цифр, при этом начинаться должны с буквы. Также в них недопустимы пробелы и любые специсмволы, кроме знака подчеркивания.

Только после сохранения всех составных частей, будет предложено сохранить собственно файл проекта. Назовем его "first". После сохранения можно, наконец-то скомпилировать исполняемый файл, нажав Ctrl+F9 (или Project Make), закрыть Delphi и посмотреть, что мы имеем в папке Project1. А в ней, как и было обещано, будет файл main.pas - программный код для формы, main.dfm - описание формы, first.dpr - сам проект, first.res - файл ресурсов проекта, main.dcu - подготовленный к компиляции модуль, и, разумеется, first.exe - исполняемый файл готового приложения. Так же вы обнаружите в нем все служебные файлы Delphi, в которых хранится дополнительная информация о проекте и настройках рабочей среды для него - файлы first.cfg, first.dof и first.dsk.

Чтобы теперь вернуться к работе над этим проектом, достаточно дважды щелкнуть по файлу first.dpr, в результате будет загружена и Delphi IDE, и этот проект в нее.

Типы проектов и депозитарий

Только что мы рассмотрели создание наиболее распространенного типа проекта - приложения Windows со стандартным графическим интерфейсом. Но на самом деле, возможности Delphi этим не ограничены, вы можете создавать приложения самого разнообразного характера, включая консольные (для текстового режима Windows), динамически подключаемые библиотеки (DLL), сервисы для Windows NT/2000/XP, межплатформенные приложения CLX (Delphi 6,7) или приложения для платформы Microsoft .NET (Delphi 8, 2005). Чтобы создать приложение определенного типа, следует из подменю File New выбрать пункт Other. Таким образом, откроется окно, позволяющее выбрать тип нового приложения или добавить какой-либо специфический модуль к существующему проекту (рис. 2.7).

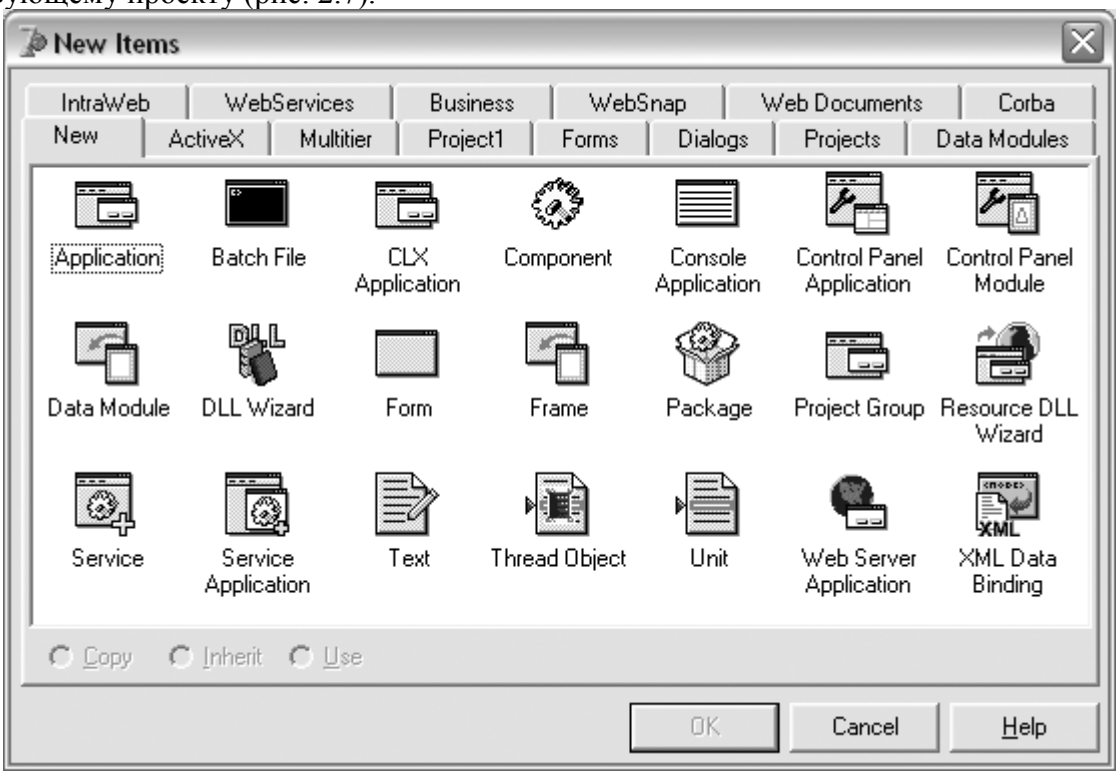


Рис. 2.7. Выбор нового приложения или модуля в Delphi 7

Выбор вариантов тут весьма обширен, причем, помимо типовых модулей и классов приложений, присутствуют различные мастера, позволяющие упростить процесс создания того или иного модуля (Wizards), а так же специализированные стандартные формы вроде диалоговых окон или окна "О программе". Рассмотрим некоторые из них подробнее, для чего пройдемся по отдельным закладкам окна New Items.

Начнем с закладки New. На ней представлены наиболее часто востребованные, по мнению разработчиков, варианты. И действительно, тут можно найти стандартное графическое Windows-приложение (Application), форму (Form), программный модуль (Unit), текстовое приложение командной строки (Console Application), и другие варианты, как-то Data Module (полезен для разработки баз данных), DLL Wizard, Component и т.д.

На закладках Forms и Dialogs моно найти ряд стандартных диалоговых окон и даже мастер по разработке диалогового окна.

Закладка Projects дает возможность начать проект того или иного типа, или даже воспользоваться мастером для создания многооконного приложения.

Чтобы создать элемент управления ActiveX или приложение для COM+, следует обратиться к шаблонам на закладке ActiveX. Ну а прочие закладки, в том числе IntraWeb, WebSnap и т.д., позволяют создавать специализированные приложения соответствующего типа или модули к ним. Их количество и названия зависят от версии Delphi и варианта поставки.

Но на самом деле, данное окно, по большому счету, подобно палитре компонент, с тем лишь исключением, что если палитра компонент являет собой представление VCL, то окно New Items - во многом является отображением депозитария (Object Repository). В депозитарии хранятся заготовки форм и иных модулей, которые вы можете многократно использовать в своих проектах. При этом для того, чтобы поместить форму в депозитарий, достаточно воспользоваться ее контекстным меню и выбрать в нем пункт Add to Repository.

Прочие средства IDE

На текущий момент мы уже рассмотрели такие составные части, предоставляемые интегрированной средой разработки Delphi, как главное окно вместе с его меню, окном выбора модулей и палитрой компонентов, и инспектора объектов. Теперь обратимся к такой важной части, как окно редактора кода. Следует отметить, что до появления графических средств разработки, подобных Delphi, еще во времена MS-DOS и ранних версий Windows, IDE для программирования как раз и состояли из редактора кода да самого компилятора. Таким образом, редактор кода - это наиболее характерный и устоявшийся элемент в любой среде разработки приложений.

Применительно ко всем современным версиям Delphi, редактор кода имеет все стандартные возможности редактирования текста (вроде работы с буфером обмена), а так же ряд особенностей, характерных для редакторов кода, а именно:

Редактор всегда работает с моноширинным шрифтом (т.е. все буквы имеют одинаковую ширину). Это необходимо, поскольку в противном случае было бы тяжело ориентироваться в коде программы;

Моноширинный режим позволяет использовать такой метод, как колоночную разбивку. Иначе говоря, копировать и перемещать можно не только отдельные слова или строки, но и вырезать, копировать и вставлять прямоугольные фрагменты текста;

Редактор постоянно отображает позицию курсора, т.е. в какой строке и колонке находится точка ввода;

Отсутствие автоматического переноса строк. Поскольку в программировании каждый символ, включая обрыв строки, что-то значит, то чтобы программисты не гадали, где в коде конец строки, а где автоматический перенос, такого режима правки нет в принципе. К тому же это мешало бы ориентироваться в номерах строк;

Подсветка синтаксиса выделяет ключевые слова и прочие специфические языковые конструкции;

При перемещении по тексту стрелкой вправо по окончании текста в строке, курсор не переносится на сроку вниз, а продолжает перемещаться дальше;

Вы можете устанавливать закладки, т.е. помечать место в тексте при помощи сочетания горячих клавиш и быстро перемещаться между ними;

Также предусмотрены такие функции, как автоподстановка кода по ключевым фразам, автозавершение кода и т.д.

При всем этом следует отметить, что практически все функции редактора, включая правила подсветки, сочетания горячих клавиш, поведение курсора, автоподстановка и т.д., гибко настраиваются. Для этого следует открыть окно свойств редактора (Tools Editor options) и настроить параметры на свое усмотрение. Впрочем, если у вас еще не сложились какие-то предпочтения в работе с подобными редакторами, то можно этого и не делать - предлагаемые изначально настройки вполне удобны.

Что касается внешнего вида окна редактора, то в нем, при стандартных параметрах IDE, помимо области редактора кода имеется еще и проводник кода, упрощающего процесс навигации по файлу (рис. 2.8).

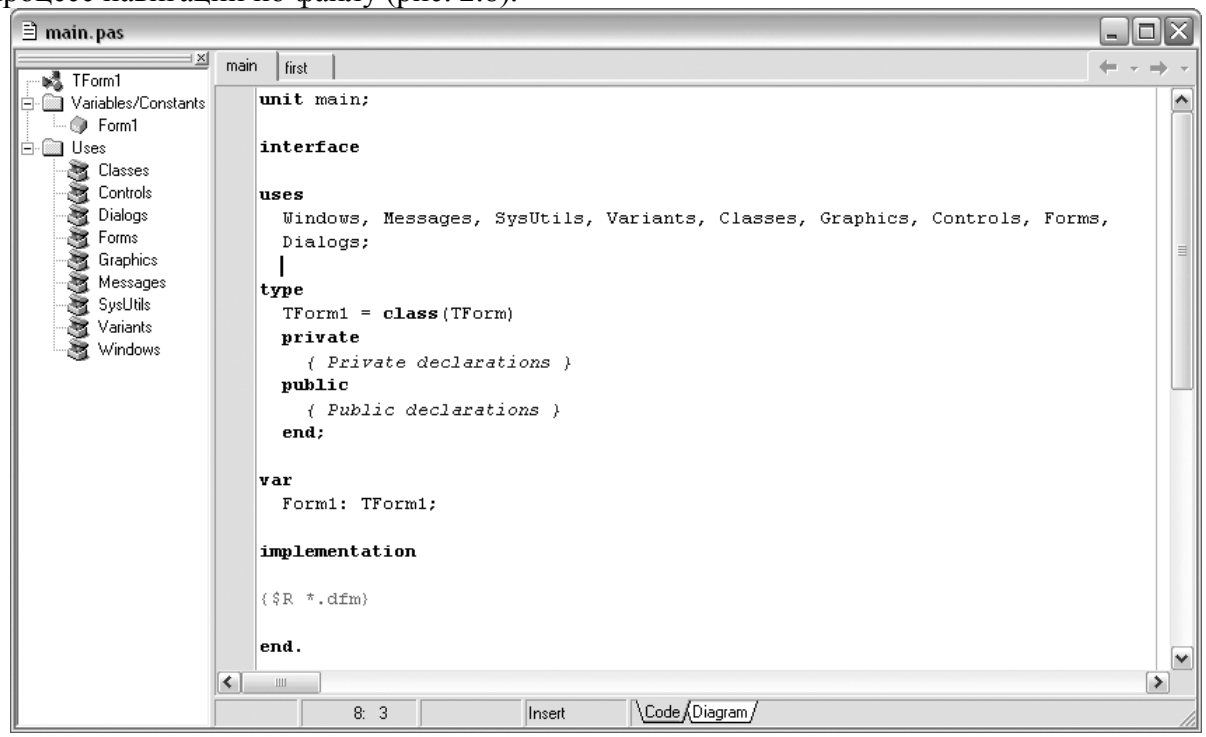


Рис. 2.8. Окно редактора кода с проводником и загруженным файлом новой формы

Обратите внимание на то, что весь программный код файла main.pas, показанный в этом окне, был создан автоматически, равно как и код для файла проекта - first.dpr. Иначе говоря, всю работу по созданию базовых блоков приложения интегрированная среда Delphi делает за вас.

Завершая тему окна редактора, отметим, что для того, чтобы загрузить в него какой-либо произвольный файл, следует воспользоваться главным меню (File Open). При открытии проекта файлы открытых форм загружаются в него автоматически, а чтобы загрузить в него иные файл проекта, используйте кнопку View Unit (сочетание горячих клавиш - Ctrl+F12) на инструментальной панели View. Если же требуется загрузить как исходный код, так и саму форму, то воспользуйтесь находящейся по соседству кнопкой View Form (Shift+F12).

Ну и последнее не рассмотренное нами окно - Object TreeView, или окно дерева объектов, служит для просмотра иерархии элементов управления (кнопок, переключателей и т.п.) относительно формы. Т.е., если проводник кода упрощает процесс поиска того или иного компонента в исходном коде программы, то дерево объектов поможет быстрее сориентироваться в элементах, находящихся на форме.

Создание приложения командной строки

Итак, мы уже ознакомились со всеми основными функциями IDE Delphi, и даже создали простейшее приложение для Windows. Однако целью данной части книги является все-таки изучения основы основ - языка Object Pascal. Поэтому для того, чтобы не отвлекаться по ходу его изучения на второстепенные (по отношению к самим языковым конструкциям) детали, рассмотрим вариант создания консольного приложения, т.е. фактически, программы для DOS. Не стоит думать, что это будет шагом назад, или что это давно морально устарело. На самом деле, абсолютно все правила языка действуют совершенно одинаково для любых программ, будь они под DOS, Windows, .NET, или Linux. Вместе с тем, отсутствие необходимости в параллельном изучении специфических для конкретной платформы особенностей (а уж тем более - в параллельном изучении такой обширной библиотеки, как VCL!) существенно упростит нашу задачу. Более того, консольные приложения в типичном случае могут состоять всего лишь из одного единственного файла, что так же упрощает понимание предмета изучения, коим на настоящий момент является сам язык программирования. Ну а после того, как будет выучен язык, применить его для создания полноценных Windows-приложений не составит труда, да и изучение VCL станет легче ввиду того, что будет ясно, что, как и почему в ней устроено.

После такого небольшого отступления, приступим к созданию первого консольного приложения в среде Delphi. Для этого откройте окно New Items (File New Other) и на закладке New дважды щелкните по значку Console Application. В результате откроется окно редактора с загруженным в него проектом (листинг 2.1).

Листинг 2.1. Заготовка приложения командной строки

```
program Project1; {$APPTYPE CONSOLE} uses SysUtils; begin { TODO -oUser -cConsole Main : Insert code here } end.
```

Первой строкой идет название программы, в данном случае это Project1, затем Delphi вставило для себя указание, что это - приложение для командной строки, после чего следует ключевое слово "uses" и перечисление необходимых дополнительных файлов (sysutils), ну и после этого, со слова begin начинается собственно тело программы. Завершается любая программа в Pascal ключевым словом end с точкой. Между ключевыми словами begin и end, в фигурных скобках, вставлен автоматический комментарий, не влияющий на выполнение программы, так что при желании можно его удалить.

А теперь самостоятельно напишем первую программу в Delphi! По традиции, она у нас будет выводить фразу "Hello, World!". Для этого на том месте, где находился комментарий, напишем одну строчку кода:

```
write(Hello, World!);
```

Все! Теперь можно сохранить и скомпилировать нашу программу. Для этого щелкните по кнопке Save или Save All на стандартной панели инструментов, в качестве пути к файлу укажите какой-либо каталог на вашем жестком диске (например, создайте папку HelloWorld на диске C:), а саму программу можно назвать hello. Таким образом, все наше приложение будет сохранено, а исходный код примет вид, показанный в листинге 2.2.

Листинг 2.2. Программа hello

```
program hello; {$APPTYPE CONSOLE} uses SysUtils; begin write(Hello, World!); end.
```

Обратите внимание, что название изменилось автоматически. Теперь остается получить исполняемый (exe) файл, для чего скомпилируем программу, нажав Ctrl+F9. Теперь запустим программу. Поскольку она у нас рассчитана на режим командной строки, то для начала откроем командную строку (Пуск > Все программы > Стандартные > Командная строка в Windows XP, или Пуск > Программы > Стандартные > Сеанс MS-DOS в Windows 98). В командной строке вызовем нашу программу, не забыв указать

полный путь к ней. Например, если вы сохранили проект в C:\HelloWorld, а саму программу назвали Hello, то и путь укажите соответствующий, т.е.:

C:\HelloWorld\hello.exe

Запустив программу (т.е. введя в командной строке путь и нажав Enter), вы сразу же увидите результат ее выполнения - она выведет фразу "Hello, World!", и завершится. Собственно говоря, именно по этой причине мы сначала открыли командную строку, а лишь затем запустили программу, поскольку непосредственный ее запуск (через Проводник Windows, или прямо из Delphi - по F9), привел бы к тому, что на экране просто мелькнул бы автоматически запущенный сеанс MS-DOS и сразу закрылся бы. Это объясняется тем, что программа завершает свою работу сразу после того, как будет достигнут ее конец, обозначенный как "end.", а это в нашем случае произойдет моментально после вывода текста. Но мы можем изменить такое ее поведение, добавив еще одну сроку кода непосредственно после "write(Hello, World!);", которая будет дожидаться момента, пока пользователь не нажмет клавишу Enter. Выглядеть она будет так:

readln;

Теперь можно запустить наше приложение прямо из среды Delphi, нажав F9. В результате можно будет наблюдать надпись "Hello, World!" на фоне черного окна автоматически запущенного сеанса консоли командной строки до тех пор, пока вы не нажмете Enter.