

Часть 3 Создание приложений Windows

МГТУ им. Н.Э. Баумана

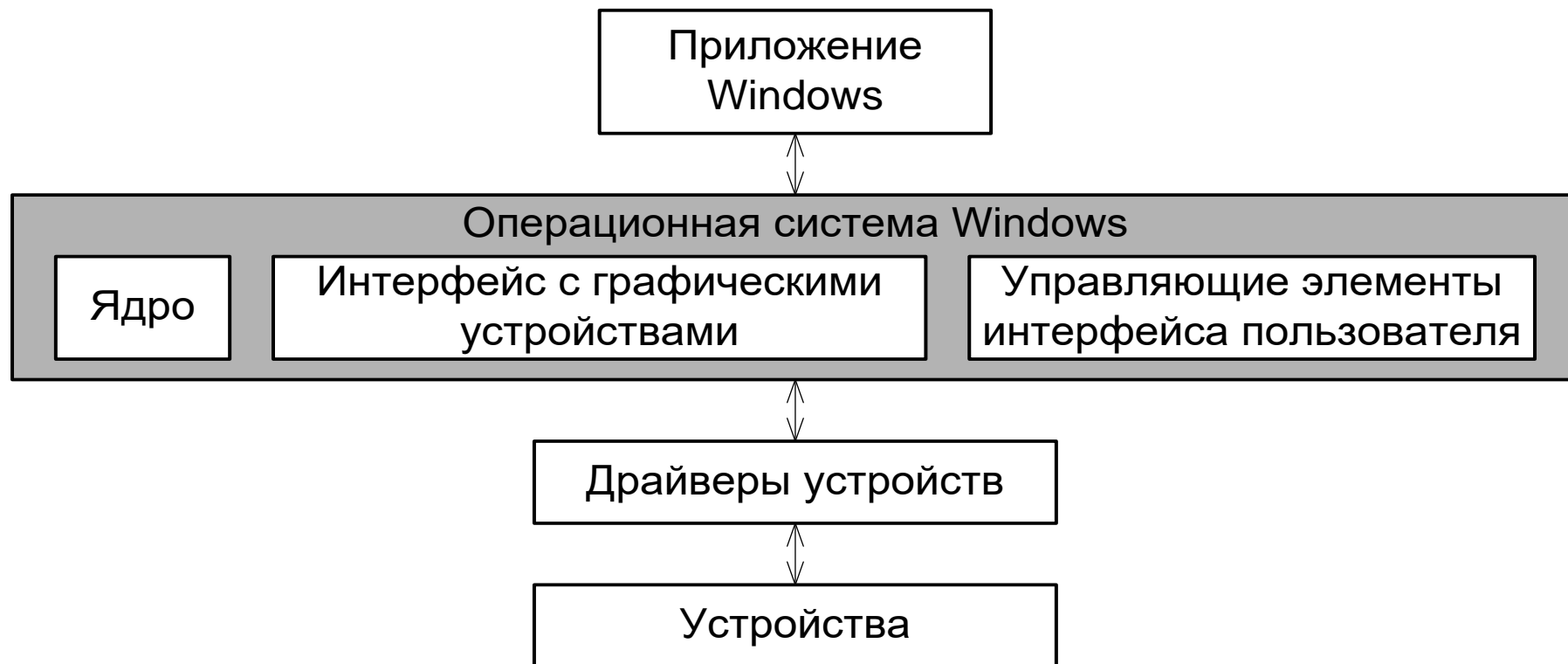
Факультет Информатика и системы
управления

Кафедра Компьютерные системы и сети

Лектор: д.т.н., проф.

Иванова Галина Сергеевна

Введение. Особенности программирования «под Windows»



Управление техническими средствами осуществляется через **API** (*Application Program Interface*) – набор из нескольких тысяч функций, выполняющих все системно-зависимые действия, такие как выделение памяти, вывод на экран и т.д.

Принцип событийного управления

Приложение (в отличие от программы) – набор подпрограмм, вызываемых при наступлении некоторого *события*, которым считается любое изменение в системе, касающееся данного приложения.

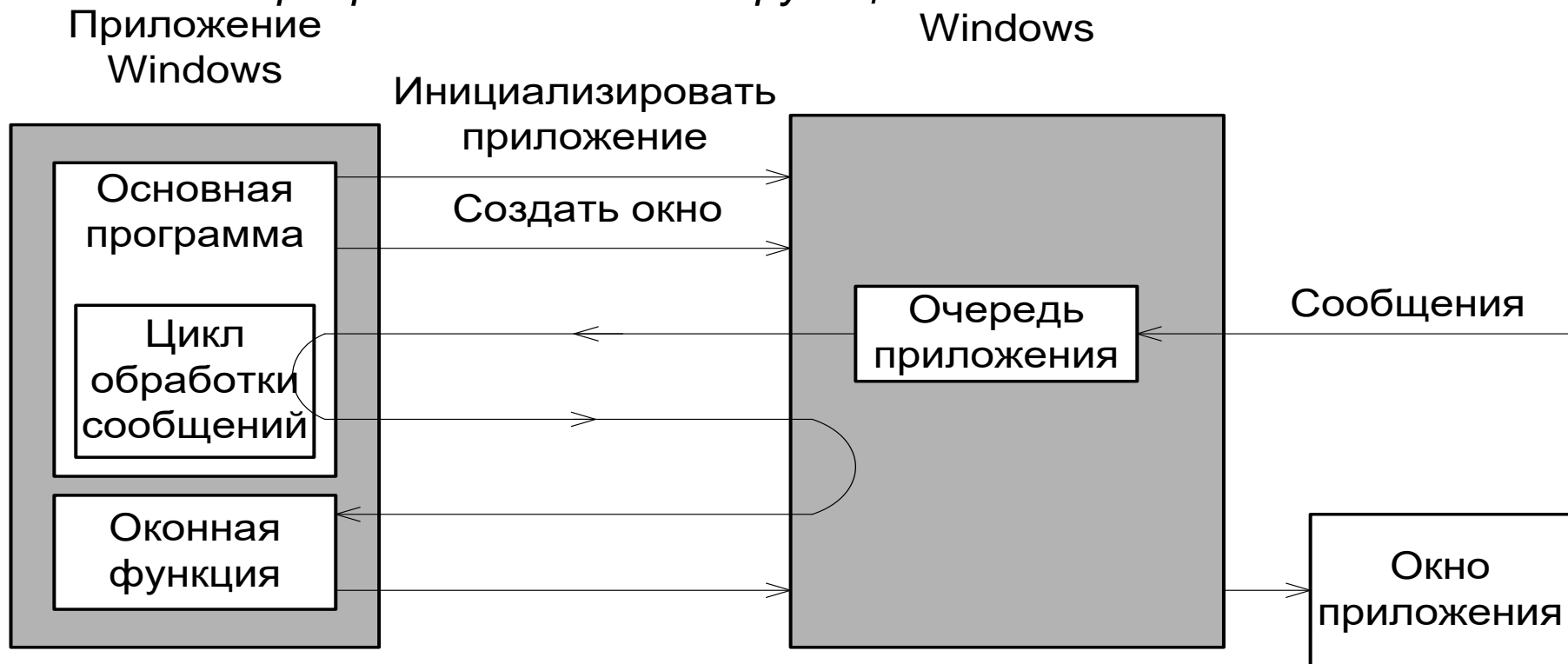
Каждому приложению на экране соответствует окно.



Окно – самостоятельно существующий объект, параметры которого хранятся в специальной структуре данных, а поведение определяется обработчиками сообщений, составляющими **оконную функцию**.³

Структура приложения

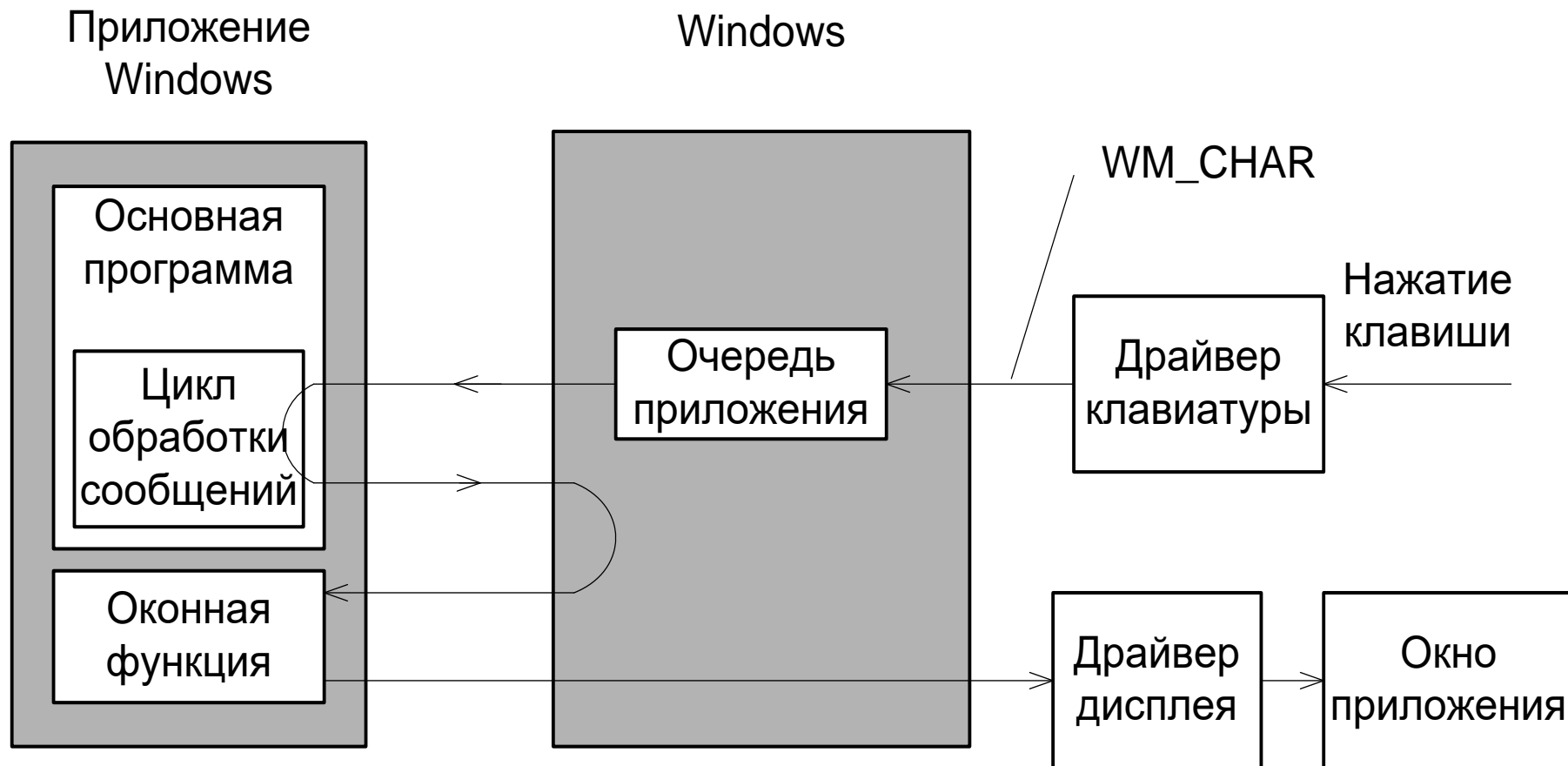
Минимально любое приложение Windows состоит из двух частей:
основной программы и оконной функции.



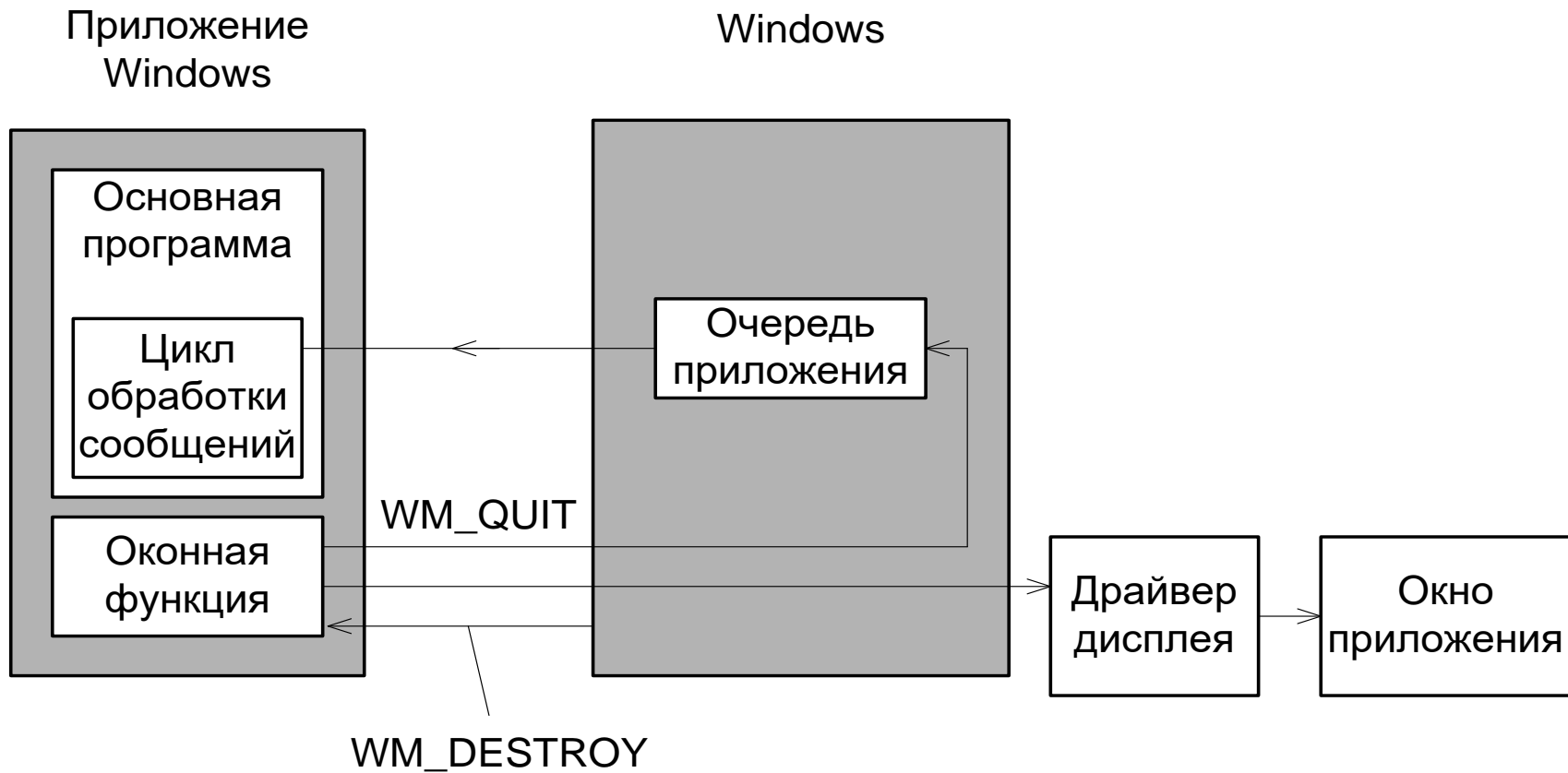
Появившиеся в очереди сообщения выбираются циклом обработки сообщений и передаются *через Windows* соответствующей оконной функции приложения.

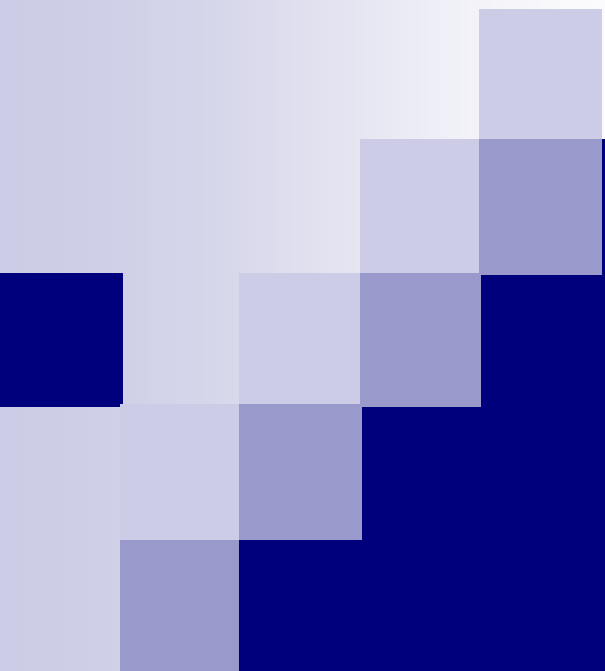
Для выполнения действий обработчики сообщений обращаются к функциям API.

Обработка сообщения от клавиатуры



Завершение приложения





Глава 8

Программирование в среде Delphi с использованием библиотеки VCL

МГТУ им. Н.Э. Баумана

Факультет Информатика и системы
управления

Кафедра Компьютерные системы и сети

Лектор: д.т.н., проф.

Иванова Галина Сергеевна

8.1 События Delphi и их обработчики

Обработчики сообщений Windows предусмотрены у объектов класса TForm и классов управляющих компонентов, таких как кнопки, редакторы и т. п.

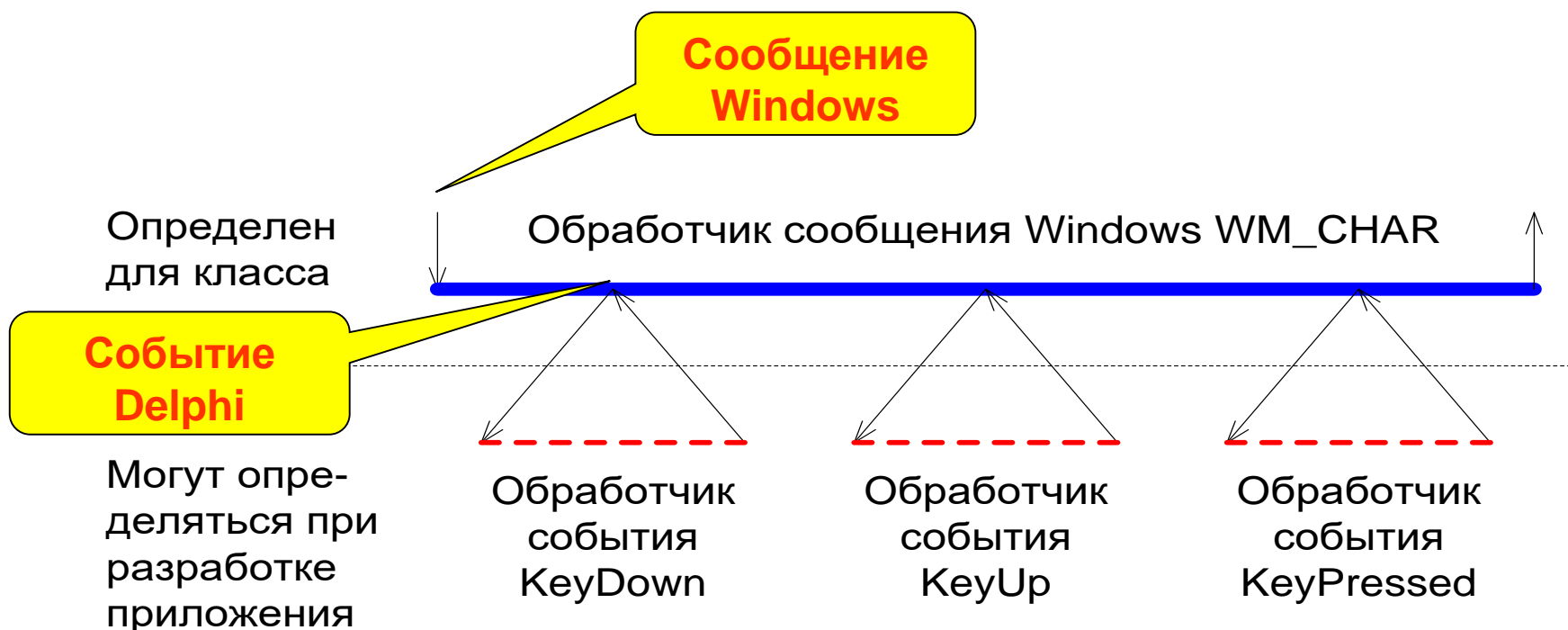
Обработка выполняется следующим образом:

1. В системе происходит событие, например, пользователь передвинул мышь или нажал на клавишу клавиатуры, в результате генерируется сообщение об этом событии – *сообщение Windows*.
2. Сообщение Windows диспетчируется конкретному приложению.
3. В приложении сообщение передается активному компоненту (окну или управляющему элементу).
4. Метод обработки сообщения Windows компонента инициирует заранее предусмотренные *события Delphi*.
5. Если в приложении предусмотрен соответствующий обработчик события Delphi, то он вызывается, если нет – то продолжается обработка сообщения Windows.

События Delphi

Обработчики сообщений Windows, встроенные в классы компонентов VCL, инициируют множество **событий Delphi**.

Например, обработчик события клавиатуры WM_CHAR класса TForm инициирует три события Delphi.



Обработчики событий Delphi

Для каждого обработчика событий предусмотрен заголовок и определенный список передаваемых ему параметров.

Имя компонента

Имя события Delphi

а) `procedure TForm1.FormActivate (Sender: TObject) ;`

Параметр **Sender** – отправитель (инициатор события).

б) `procedure TForm1.Edit1KeyPress (Sender: TObject;
var Key: Char) ;`

Параметр **Key** – символ ANSI.

в) `procedure TForm1.Edit1KeyDown (Sender: TObject;
var Key: Word; Shift: TShiftState) ;`

Параметры: **Key** – виртуальный код, **Shift** – управляющие клав.

г) `procedure TForm1.Edit1KeyUp (Sender: TObject;
var Key: Word; Shift: TShiftState) ;`

8.2 Класс формы TForm

Основное окно приложения строится на базе класса **TForm**.

При входе в Delphi предоставляется заготовка приложения, которая «умеет» (содержит определенные обработчики сообщений Windows) выполнять стандартные действия.

Свойства:

Caption – заголовок окна (по умолчанию Form1);

FormStyle – вид формы(обычное, родительское, дочернее, неперекрываемое);

Width, Height, Top, Left – размеры и местоположение;

Color – цвет фона;

Font – характеристики шрифта;

Cursor – форма курсора мыши и т. д.

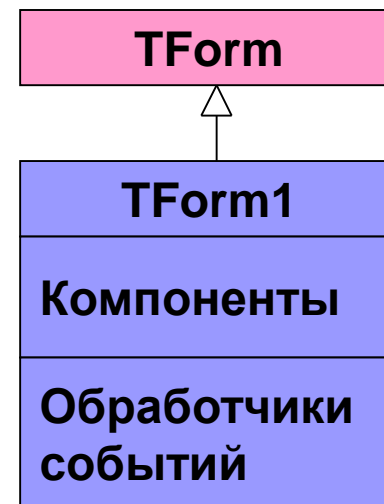
Методы:

Show – показать;

Hide – спрятать;

Close – закрыть;

ShowModal – показать в модальном режиме и т. д.



События Delphi, обрабатываемые объектами класса TForm

а) *при изменении состояния формы:*

OnCreate – в начальной стадии создания формы - используется при необходимости задания параметров (цвет или размер);

OnActivate – при получении формой фокуса ввода (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 (окно становится неактивным);

OnHide – при удалении формы с экрана (окно становится невидимым);

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

OnClose – при закрытии формы;

OnDestroy – при уничтожении формы;

События Delphi для класса TForm (2)

б) от клавиатуры и мыши:

OnKeyPressed – при нажатии клавиш, которым соответствует код ANSI;

OnKeyDown, OnKeyUp – при нажатии и отпускании любых клавиш;

OnClick, OnDbClick – при обычном и двойном нажатии клавиш мыши;

OnMouseMove – при перемещении мыши (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 компонент.

8.3 Основные визуальные компоненты и средства визуализации сообщений пользователю

1. Метка (класс *TLabel*)

Метка – окно с текстом и может использоваться для формирования на форме некоторых надписей или подписей.



Label1

Основные свойства:

Caption – заголовок – текст, выводимый в окне компонента.

Name – имя компонента в программе.

Visible – определяет видимость компонента.

Alignment – определяет способ выравнивания текста в окне компонента: *taCenter* - по центру; *taLeftJustify* - по левой границе; *taRightJustify* - по правой границе.

Font – определяет шрифт текста.

Color – определяет цвета текста и фона в окне.

WordWrap – определяет, разбивать или нет текст на строки.

Transparent – определяет, виден ли рисунок фона через окно.

Основные визуальные компоненты (2)

2. Строчный редактор (класс *TEdit*)

Компонент представляет собой окно, обычно выделенное цветом, которое может использоваться, например, для организации ввода информации.

Основные свойства:

Text – строка текста.

ReadOnly – определяет возможность ввода информации в окно.

Name – имя компонента в программе.

Visible – определяет видимость компонента.

Enable – доступность компонента.

Основные методы:

Clear – очистка поля Text.

SetFocus – установка фокуса ввода.

Основные события:

OnKeyPressed – нажатие алфавитно-цифровой клавиши.



Основные визуальные компоненты (3)

3. Кнопка

Компонент представляет собой окно, в котором размещается название кнопки. Используется для инициирования каких-либо действий.

Основные свойства:

Caption - название кнопки.

Name – имя компонента в программе.

Visible – определяет видимость компонента.

Enable – доступность компонента.

Default – определяет, генерируется ли событие *OnClick* для данной кнопки при нажатии клавиши *Enter*.

Cancel – аналогично, но для клавиши *Esc*.

Основные события:

OnClick – «щелчок» мышкой в зоне компонента.

Основные методы:

SetFocus – установка фокуса ввода.



Средства визуализации сообщений ПОЛЬЗОВАТЕЛЮ

1 Метод MessageBox (класс TApplication)

```
function MessageBox(Text, Caption: PChar;  
    Flags: Longint=MB_OK): Integer;
```

Пример:

```
Application.MessageBox('Строка пуста',  
    'Error', mb_Ok);
```

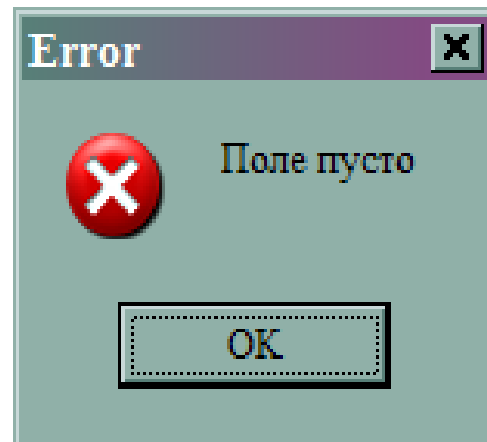
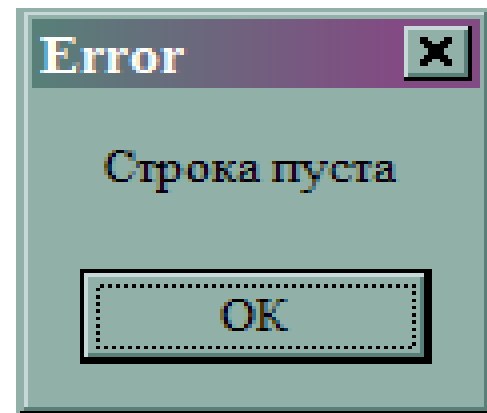
Возвращает код ответа.

2 Функция MessageDlg

```
function MessageDlg(const Msg: string;  
    DlgType: TMsgDlgType;  
    Buttons: TMsgDlgButtons;  
    HelpCtx: Longint): Word;
```

Пример:

```
MessageDlg('Поле пусто', mtError,  
    [mbOk], 0);
```



Вычисление квадрата заданного числа



Form1:

Name:=MainForm;
Caption:='Возведение в квадрат';

Label1:

Name:='InputLabel';
Caption:='Введите число';

Label2:

Name:=OutPutLabel;
Caption:='Квадрат числа равен:';

Edit1:

Name:=InputEdit;

Edit2:

Name:=OutPutEdit;
Enable:=false;
ReadOnly:=true;

Button1:

Name:=NextButton;
Caption:='Следующее';

Button2:

Name:=ExitButton;
Caption:='Выход';

Формы интерфейса

Введите значение:

Следующее

Выход

Введите значение:

67

Квадрат введенного числа равен:

4489

Следующее

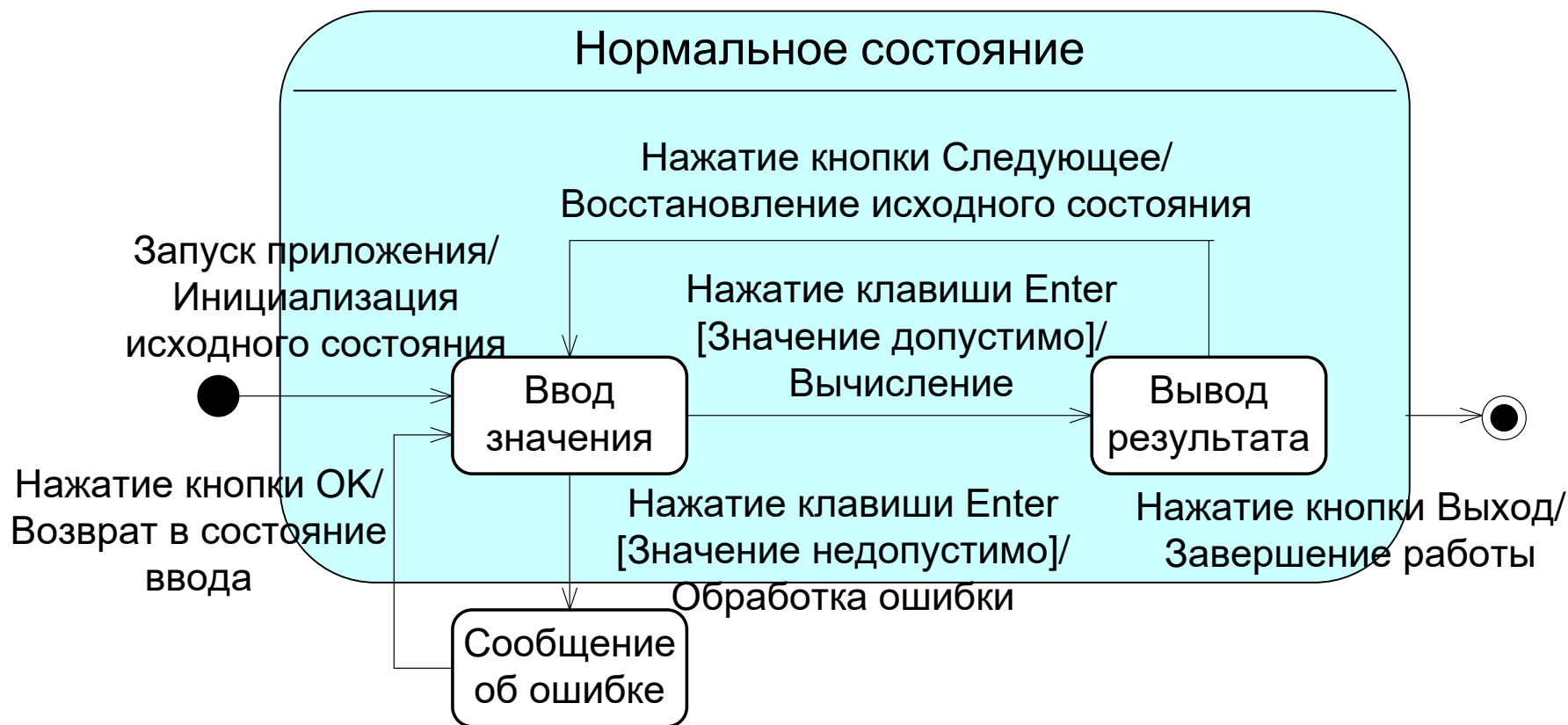
Выход

Error

Введенная строка содержит лишние символы.

OK

Диаграмма состояний интерфейса

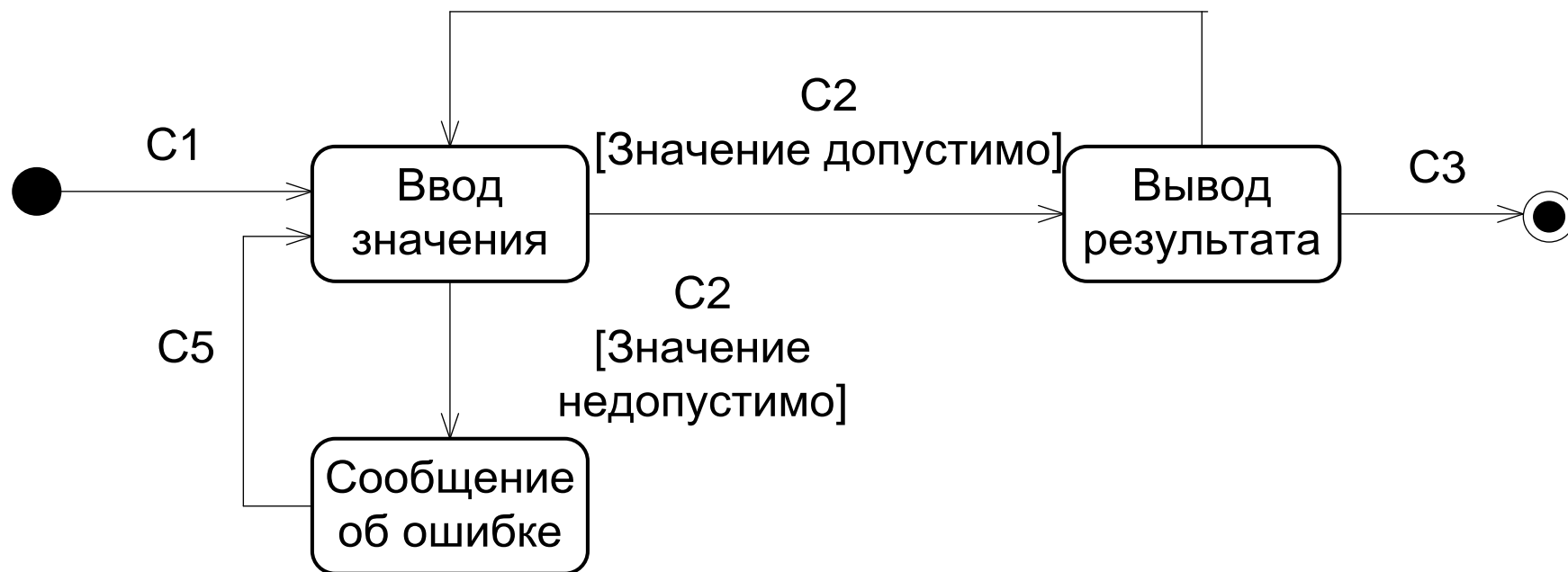


Объектная декомпозиция (диаграмма объектов)



Определение обрабатываемых событий по диаграмме состояний интерфейса

C4



C1 – активация формы;

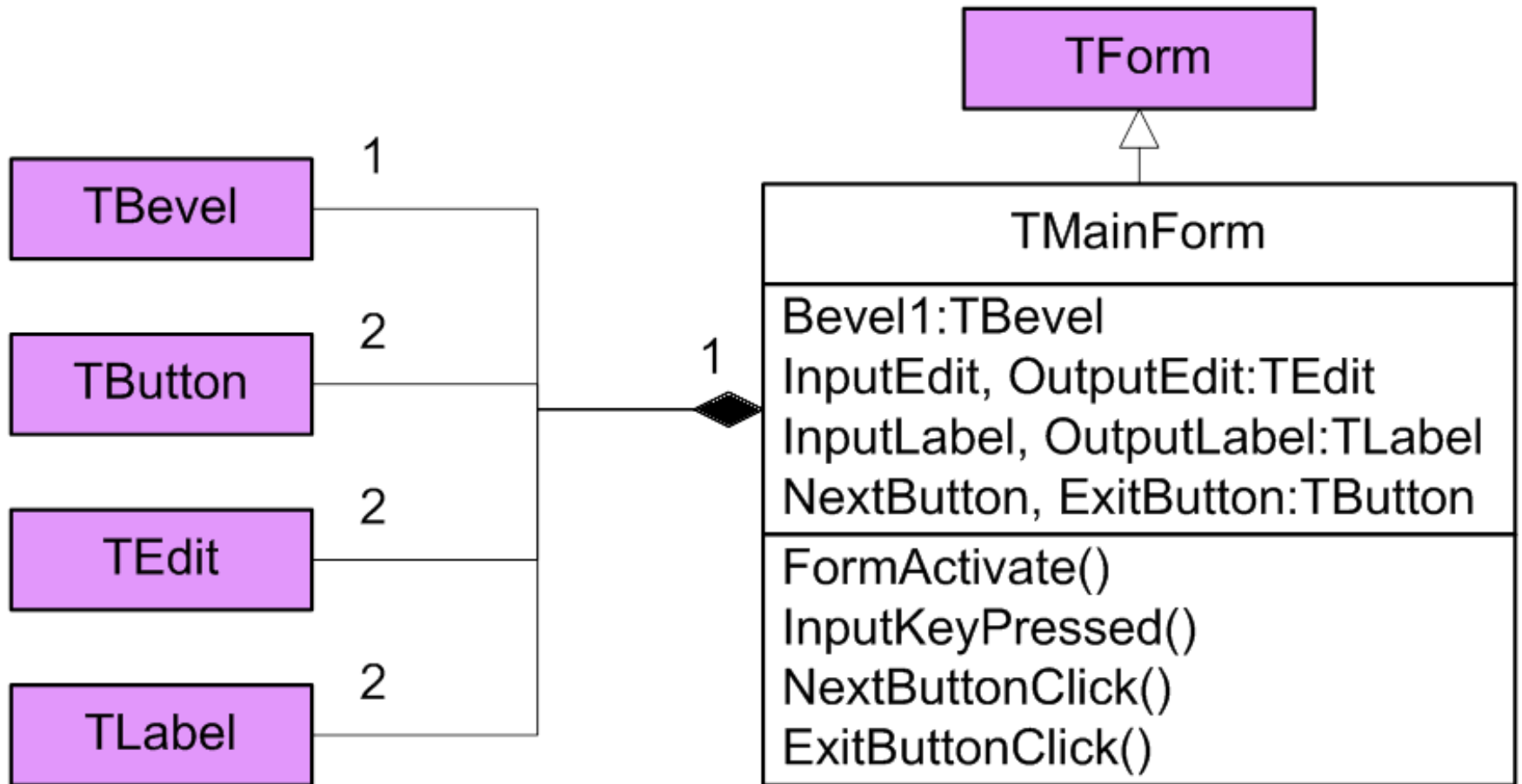
C2 – ввод Enter;

C3 – нажатие кнопки Выход;

C4 – нажатие кнопки Следующее;

C5 – нажатие кнопки ОК

Диаграмма классов приложения



Файл проекта

```
Program Ex8_01;
```

```
Uses
```

```
    Forms,
```

```
    MainUnit in 'MainUnit.pas' {MainForm};
```

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

```
begin
```

```
    Application.Initialize;
```

```
    Application.CreateForm(TMainForm, MainForm);
```

```
    Application.Run;
```

```
end.
```


Файл MainUnit

```
Unit MainUnit;
interface
  Uses Windows, Messages, SysUtils, Classes, Graphics,
        Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
  Type TMainForm = class(TForm)
    Bevel1: TBevel;
    NextButton, ExitButton: TButton;
    InputLabel, OutPutLabel: TLabel;
    InPutEdit, OutPutEdit: TEdit;
    procedure FormActivate(Sender: TObject);
    procedure InPutEditKeyPress(Sender: TObject;
              var Key: Char);
    procedure NextButtonClick(Sender: TObject);
    procedure ExitButtonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Файл MainUnit (2)

```
var MainForm: TMainForm;  
implementation  
{ $R *.DFM }  
procedure TMainForm.FormActivate(Sender: TObject);  
begin  
    NextButton.Enabled:=false;  
    InPutEdit.ReadOnly:=false;  
    InPutEdit.Clear;  
    InPutEdit.Enabled:=true;  
    InPutEdit.SetFocus;  
    OutPutLabel.Visible:=false;  
    OutPutEdit.Visible:=false;  
end;
```

Файл MainUnit (3)

```
procedure TMainForm.InPutEditKeyPress (Sender: TObject;  
                                         var Key: Char);  
  
var x:real;Code:integer;  
begin  
    If Key=#13 then  
        begin  
            Key:=#0;      Val (InPutEdit.Text,x,Code) ;  
            if Code=0 then  
                begin  
                    InputEdit.ReadOnly:=true;  
                    InputEdit.Enabled:=false;  
                    OutPutLabel.Visible:=true;  
                    OutPutEdit.Visible:=true;  
                    OutPutEdit.Text:=floattostr (sqr (x) ) ;  
                    NextButton.Enabled:=true;  
                    NextButton.SetFocus;  
                end  
            end  
        end  
end
```

Файл MainUnit (4)

```
else
begin
    MessageDlg('Недопустимые символы.', mtError,
               [mbOk], 0);
end
end
end;

procedure TMainForm.NextButtonClick(Sender: TObject);
begin FormActivate(NextButton); end;

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

end.
```

8.4 Расширение Delphi Pascal

Дополнительные скалярные типы данных

а) целый

Cardinal 0..2147483647 без знака 4 байта

б) логические

ByteBool	1 байт	}	true – любое число, кроме 0
WordBool	2 байта		
LongBool	4 байта		

в) СИМВОЛЬНЫЕ

ANSIChar 1 байт (ANSI)

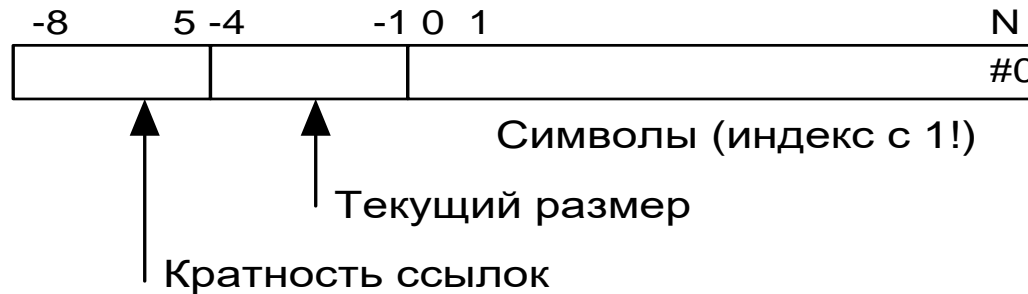
WideChar 2 байта (Unicode)

г) вещественные – денежный:

Currency -922337203685477.5808.. 922337203685477.5807 8 байт

Строковые типы

- а) **ShortString** – до 255 символов, 0-й байт – длина строки;
- б) **AnsiString** – указатель на строку ANSI символов,
WideString – указатель на строку символов Unicode:



в) **String**: {\$H+} == **ANSIString**; {\$H-} == **ShortString**;

Исключение! **String**[<Максимальная длина>] – всегда **ShortString**

Для перечисленных строк определены операции конкатенации и сравнения, а также специальные функции в **System** и **SysUtils**

г) **PChar** – указатель на массив символов, завершающийся нулем.
Строки совместимы:

<строка, кроме **PChar**> := <любая строка или символ>

<строка **PChar**> := <строковая константа или **PChar** (строка)>

Дополнительные процедуры и функции

- 1) `function ANSILowerCase (const S:String) :String` – заменяет прописные буквы на строчные;
- 2) `function ANSIUpperCase (const S:String) :String` – заменяет строчные буквы на прописные;
- 3) `function StrToInt (S:String) :Integer` – преобразует строку в целое число;
- 4) `function StrToIntDef (S:String; Default:Integer) :Integer` – то же + возвращает код ошибки;
- 5) `function StrToIntRange (S:String; Min..Max:LongInt) :LongInt` – то же + генерирует исключение `ERangeError`, если не входит в диапазон;
- 6) `function StrToFloat (S:String) :Extended` – преобразует строку в вещественное число, в качестве разделителя использует символ, указанный при настройке Windows;

Дополнительные процедуры и функции (2)

- 7) `function IntToStr(V:Integer):String` – преобразует целое число в строку символов;
- 8) `function IntToHex(V:Integer; Digits:Integer):String` – преобразует целое число в строку шестнадцатеричных символов, минимальной длины Digits;
- 9) `function FloatToStr(V:Extended):String` – преобразует вещественное число в строку символов.

Файлы

1. Изменены некоторые служебные слова и названия процедур:

Text ⇒ **TextFile**

Assign ⇒ **AssignFile**

Close ⇒ **CloseFile**

Процедурный тип данных

1) При объявлении указателя на метод добавляют `of object`

Пример:

```
Type metodproc = procedure (Sender:TObject) of object;
```

2) Переменным процедурного типа можно присвоить `nil` и проверять их значение при необходимости

Пример:

```
if Assigned(p2) then p2(x,y);
```

8.5 Классы Delphi

Основная особенность: *все объекты размещаются в динамической памяти.*

Описание класса:

```
Type <Имя объявляемого класса> = class(<Имя родителя>)  
    private      <Скрытые элементы класса>  
    protected   <Защищенные элементы класса>  
    public       <Общедоступные элементы класса>  
    published    <Опубликованные элементы класса>  
    automated    <Элементы, реализующие OLE-механизм>  
end;
```

Если имя родителя не указано, то им считается класс **TObject**.

Конструктор и деструктор

Переменные класса являются *ссылками*. В отличие от указателей операция разыменования при работе с ними не используется.

Конструктор **Create** и деструктор **Destroy** класса должны содержать вызов конструктора и деструктора TObject, которые обеспечивают **выделение** и **освобождение** памяти:

```
Constructor <Имя класса>.Create;
```

```
begin
```

```
    inherited Create;
```

```
    . . .
```

```
end;
```

```
Destructor Destroy; override; {деструктор виртуальный!}
```

```
Destructor <Имя класса>.Destroy;
```

```
begin
```

```
    . . .
```

```
    inherited Destroy;
```

```
end;
```

Сравнение объектных моделей

Простая модель
Type

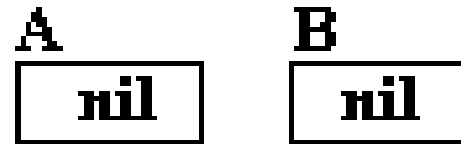
```
pTNum = ^TNum;  
TNum = Object  
    n: integer;  
    procedure Init  
        (an:integer) ;  
    end;  
Procedure TNum.Init;  
    begin  n:=an;  end;  
    .      .      .  
Var p:pTNum;  
Begin  
    New(p, Init(5));  
    WriteLn(p^.n);  
    Dispose(p);  
End.
```

Модель VCL
Type

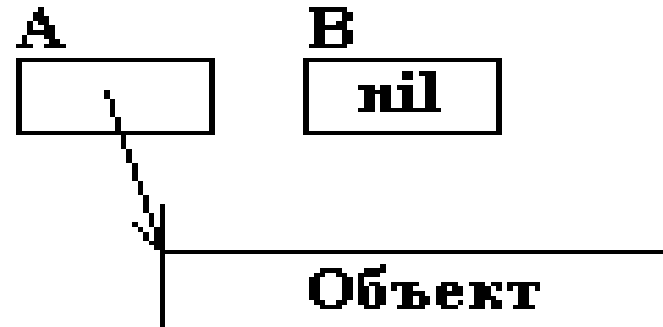
```
TNum = class  
    public  n:integer;  
    constructor Create  
        (an:integer) ;  
    end;  
Constructor TNum.Create;  
    begin  
        inherited Create;  
        n:=an;  
    end;  
    .      .      .  
Var A:TNum;  
    .      .      .  
    A:=TNum.Create(5);  
    WriteLn(A.n);  
    A.Destroy;  
    .      .      .
```

Особенности работы с объектами

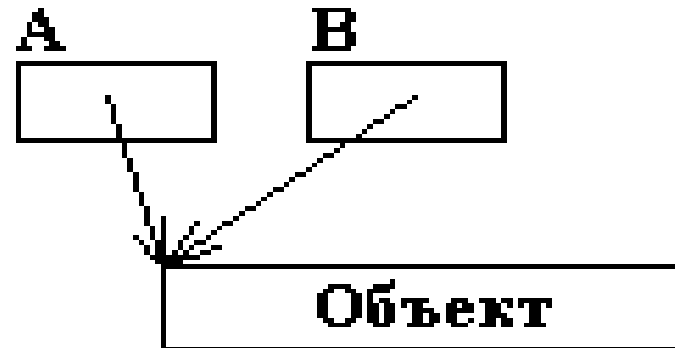
Var A, B:TNum;



A:=TNum.Create;

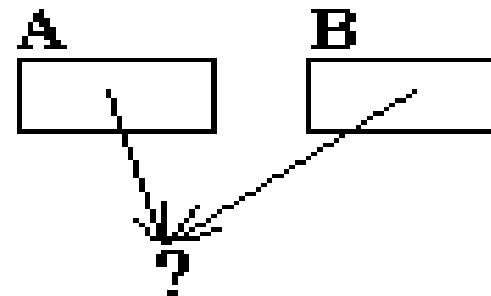


B:=A;



B.Destroy;

A.n:=3; {!!!}



Особенности переопределения методов

1) виртуальные методы:

```
procedure Print;virtual;      { в базовом класса }  
procedure Print;override;    { в производном классе }
```

2) абстрактные методы:

```
procedure Print;virtual;abstract; { в базовом класса }  
procedure Print;override;        { в производном классе }
```

Класс, содержащий абстрактные методы, называется **абстрактным**.

Объекты абстрактного класса создавать **не разрешается**.

8.6 События Мыши

1 Событие «Нажатие кнопки мыши над компонентом окна приложения»:

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

где

Button: TMouseButton - определяет, какая кнопка нажата:

```
type TMouseButton = (mbLeft, mbRight, mbMiddle); -  
соответственно, левая, правая или средняя кнопки.
```

Shift: TShiftState - определяет нажатые управляющих клавиш клавиатуры и мыши:

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

X, Y: Integer - координаты мыши относительно компонента.

События Мыши (2)

2 Событие «Движение мыши»:

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

3 Событие «Отпускание клавиши мыши»:

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

8.7 Создание графических изображений

Изображения в Windows создаются с помощью типов:

- точка,
- прямоугольник;

и 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 и прямоугольник

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;
```

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, psInsideFrame;

Mode:TPenMode – способ взаимодействия линий с фоном, например,
pmBlack – только черные линии,
pmWhite – только белые линии,
pmNor – линии не видны на фоне,
pmNot – инверсия фона и т.д..

Компонент Кисть (класс TBrush)

Объекты класса TBrush служат для заполнения внутреннего пространства.

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, например:

bsSolid, bsClear, bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal, bsCross, bsDiagCross;

BitMap:TBitMap – растровое изображение, которое будет использоваться кистью для заполнения, если свойство определено, то цвет и стиль игнорируются.

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.

Компонент Канва (класс TCanvas)

Класс создает Канву – холст для рисования.

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.

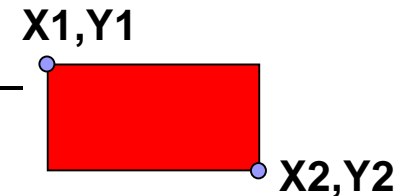
Основные методы класса TCanvas

procedure MoveTo (X, Y: Integer) – перемещает перо в указанную точку;

procedure LineTo (X, Y: Integer) – чертит линию из текущей точки в заданную;

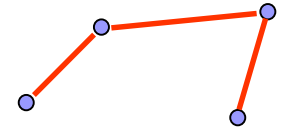


procedure Rectangle (X1, Y1, X2, Y2: Integer) – рисует и закрашивает кистью прямоугольник;

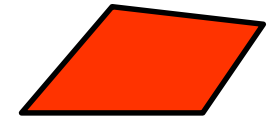


procedure Polyline (Points: array of TPoint) – рисует ломаную линию;

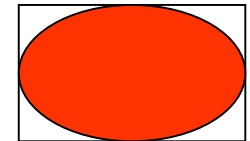
type TPoint = record X, Y: Longint; end;



procedure Polygon (Points: array of TPoint) – рисует и закрашивает кистью многоугольник;



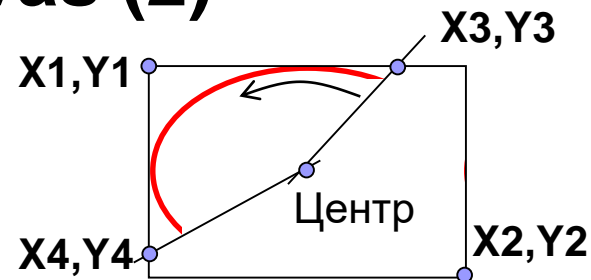
procedure Ellipse (X1, Y1, X2, Y2: Integer) – рисует эллипс в заданном прямоугольнике и закрашивает кистью;



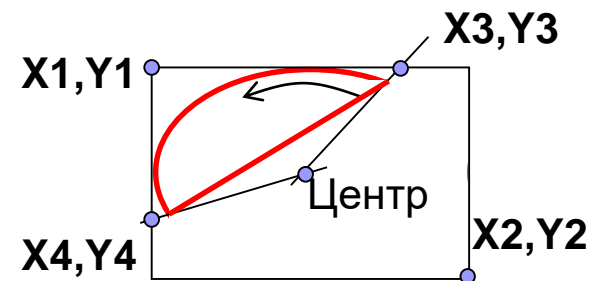
procedure FrameRect (const Rect: TRect) – очерчивает границы прямоугольника текущей кистью без заполнения;

Основные методы класса TCanvas (2)

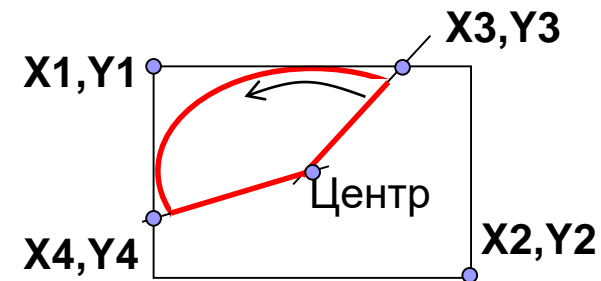
procedure Arc ($x_1, y_1, x_2, y_2, x_3, y_3,$
 $x_4, y_4: integer$) – чертит дугу эллипса в
прямоугольнике (x_1, y_1, x_2, y_2), направление –
против часовой стрелки;



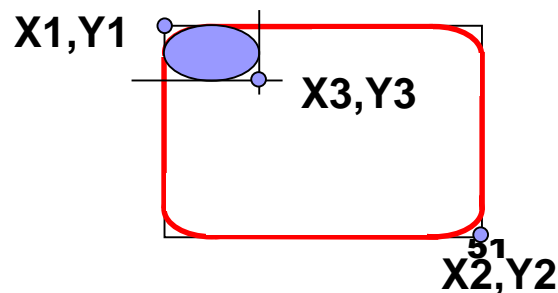
procedure Chord ($x_1, y_1, x_2, y_2, x_3, y_3,$
 $x_4, y_4: integer$) – чертит сегмент эллипса в
прямоугольнике (x_1, y_1, x_2, y_2), направление –
против часовой стрелки;



procedure Pie ($x_1, y_1, x_2, y_2, x_3, y_3,$
 $x_4, y_4: integer$) – чертит сектор эллипса в
прямоугольнике (x_1, y_1, x_2, y_2), направление –
против часовой стрелки;



procedure RoundRect ($x_1, y_1, x_2, y_2,$
 $x_3, y_3: integer$) – чертит и заполняет
прямоугольник с закругленными краями.



Основные методы класса TCanvas (3)

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);

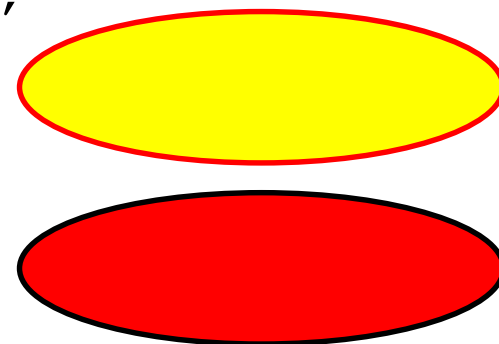
end;



procedure FloodFill(X,Y:Integer;Color:TColor; FillStyle:TFillStyle) –

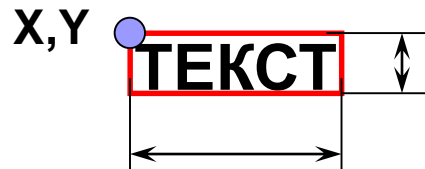
type TFillStyle = (fsSurface, fsBorder);

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



Основные методы класса TCanvas (4)

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;

Компонент Образ (класс TImage)

Образ – окно для работы с графикой.

Свойства:

AutoSize: Boolean – автоматическое изменение размера компонента в зависимости от размера картинки (при загрузке рисунка);

Stretch: Boolean – автоматическое масштабирование рисунка при изменении размеров компонента;

Picture: TPicture – свойство-объект картинка (двоичный образ, пиктограмма, графический метафайл);

класс включает методы:

procedure LoadFromFile(const FileName: string); – загрузка картинки из файла;

procedure SaveToFile(const FileName: string); – сохранение картинки в файле.

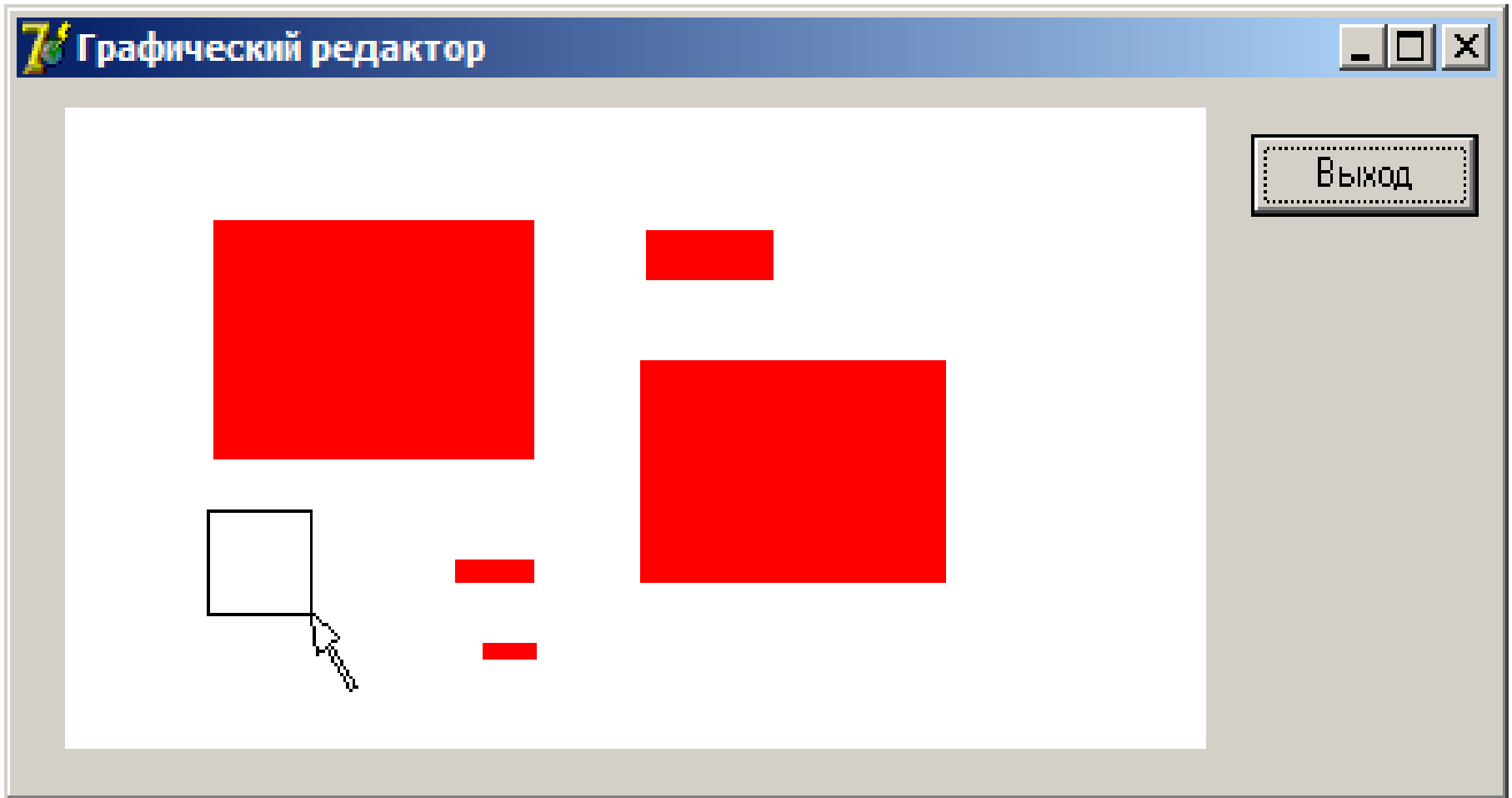
Canvas: TCanvas – холст;

Brush: TBrush – кисть;

Pen: TPen – перо;

Font: TFont – шрифт.

Рисование прямоугольников (Ex8_2)



Модуль MainUnit

```
unit MainUnit;
```

```
interface
```

```
Uses Windows, Messages, SysUtils, Classes, Graphics,  
    Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
```

```
type
```

```
    TMainForm = class(TForm)
```

```
        Image: TImage;
```

```
        ExitButton: TButton;
```

```
        procedure FormActivate(Sender: TObject);
```

```
        procedure ImageMouseDown(Sender: TObject;
```

```
            Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
```

```
        procedure ImageMouseMove(Sender: TObject;
```

```
            Shift: TShiftState; X, Y: Integer);
```

```
        procedure ImageMouseUp(Sender: TObject;
```

```
            Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
```

```
        procedure ExitButtonClick(Sender: TObject);
```

```
    end;
```

Модуль MainUnit (2)

```
var MainForm: TMainForm;  
implementation  
  Var Rect:TRect;first:boolean;  
  {$R *.DFM}  
  procedure TMainForm.FormActivate(Sender: TObject);  
    begin Image.Canvas.Brush.Color:=clWhite; end;  
  procedure TMainForm.ImageMouseDown(Sender: TObject;  
    Button:TMouseButton;Shift:TShiftState;X,Y:Integer);  
begin  
  if Button=mbLeft then  
    begin  
      Rect.Left:=x;  
      Rect.Top:=y;  
      first:=true;  
    end;  
end;  
end;
```

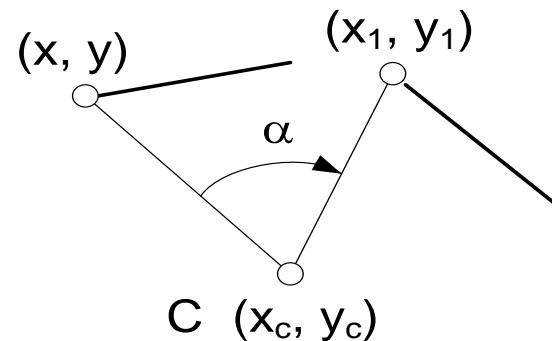
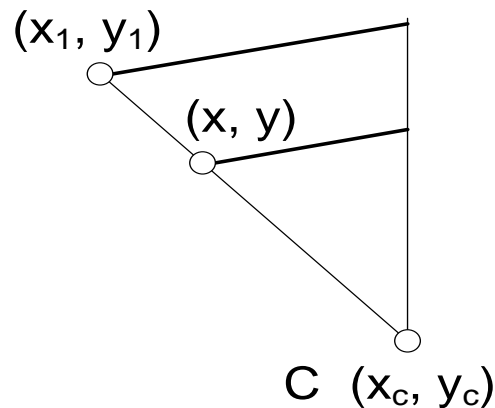
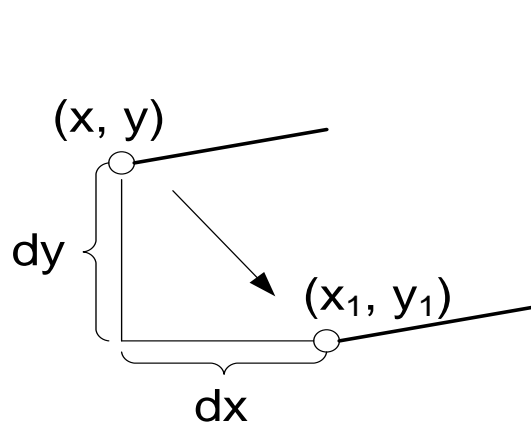
Модуль MainUnit (3)

```
procedure TMainForm.ImageMouseMove (Sender: TObject;  
    Shift: TShiftState; X, Y: Integer);  
begin  
    if ssLeft in Shift then  
        begin  
            if first then first:=not first  
            else  
                begin  
                    Image.Canvas.Pen.Color:=clWhite;  
                    Image.Canvas.Rectangle(Rect.Left,Rect.Top,  
                                            Rect.Right,Rect.Bottom) ;  
                end;  
            Rect.Right:=X;    Rect.Bottom:=Y;  
            Image.Canvas.Pen.Color:=clBlack;  
            Image.Canvas.Rectangle(Rect.Left,Rect.Top,  
                                    Rect.Right,Rect.Bottom) ;  
        end;  
    end;  
end;
```


Модуль MainUnit (4)

```
procedure TMainForm.ImageMouseUp(Sender: TObject;  
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer) ;  
begin  
    if Button=mbLeft then  
        begin  
            Image.Canvas.Pen.Color:=clWhite;  
            Image.Canvas.Rectangle(Rect.Left, Rect.Top,  
                                   Rect.Right, Rect.Bottom) ;  
            Rect.Right:=X;    Rect.Bottom:=Y;  
            Image.Canvas.Brush.Color:=clRed;  
            Image.Canvas.FillRect(Rect) ;  
            Image.Canvas.Brush.Color:=clWhite;  
            Image.Canvas.Pen.Color:=clBlack;  
        end;  
    end;  
procedure TMainForm.ExitButtonClick(Sender: TObject) ;  
begin    Close;    end;  
end.
```

8.8 Три составляющих движения объекта на плоскости



1 Перемещение

$$\begin{aligned}x_1 &= x + dx, \\y_1 &= y + dy\end{aligned}$$

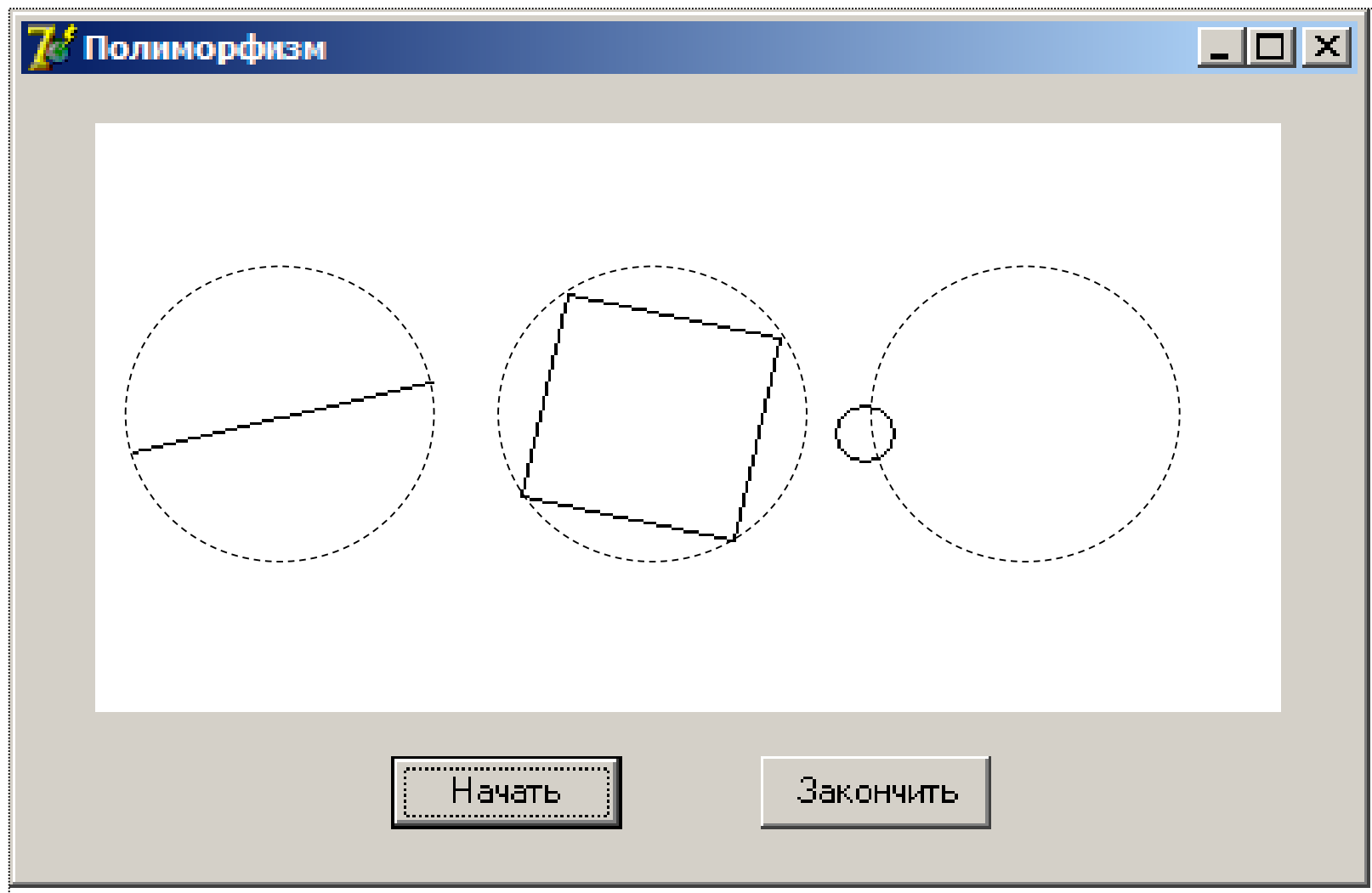
2 Масштабирование
относительно
точки C (xc, yc):

$$\begin{aligned}x_1 &= (x - x_c) * M_x + x_c, \\y_1 &= (y - y_c) * M_y + y_c\end{aligned}$$

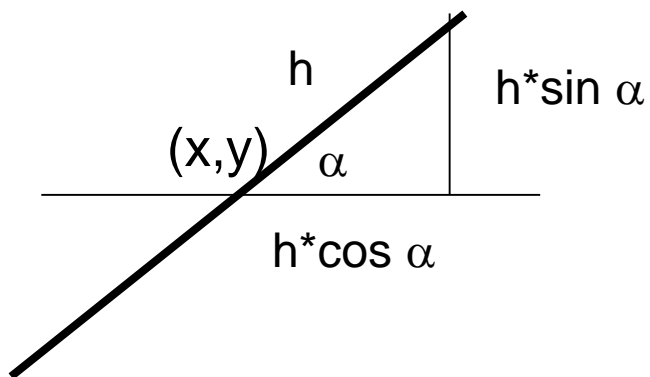
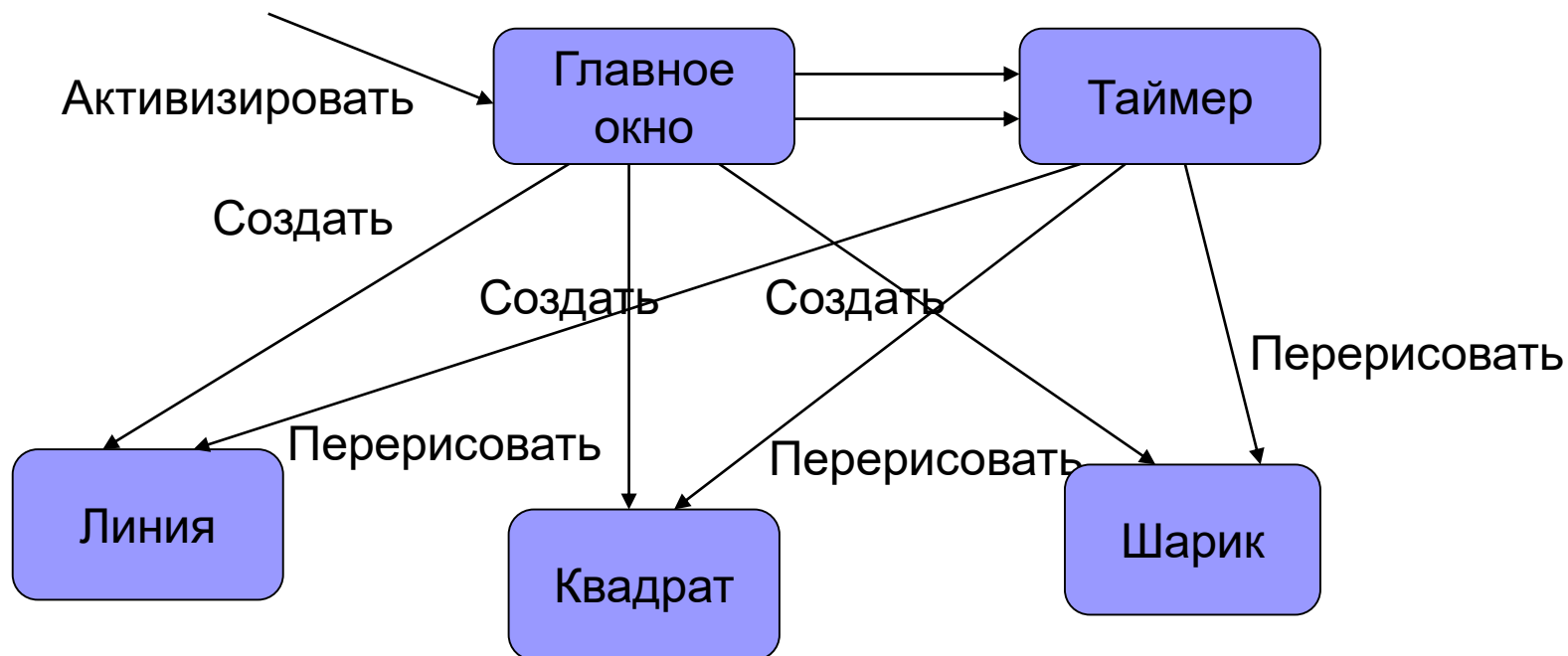
3 Поворот относительно
точки C (xc, yc):

$$\begin{aligned}x_1 &= (x - x_c) \cos \alpha + \\&\quad + (y - y_c) \sin \alpha + x_c, \\y_1 &= (y - y_c) \cos \alpha - \\&\quad - (x - x_c) \sin \alpha + y_c\end{aligned}$$

Вращение фигур (Ex8_03)



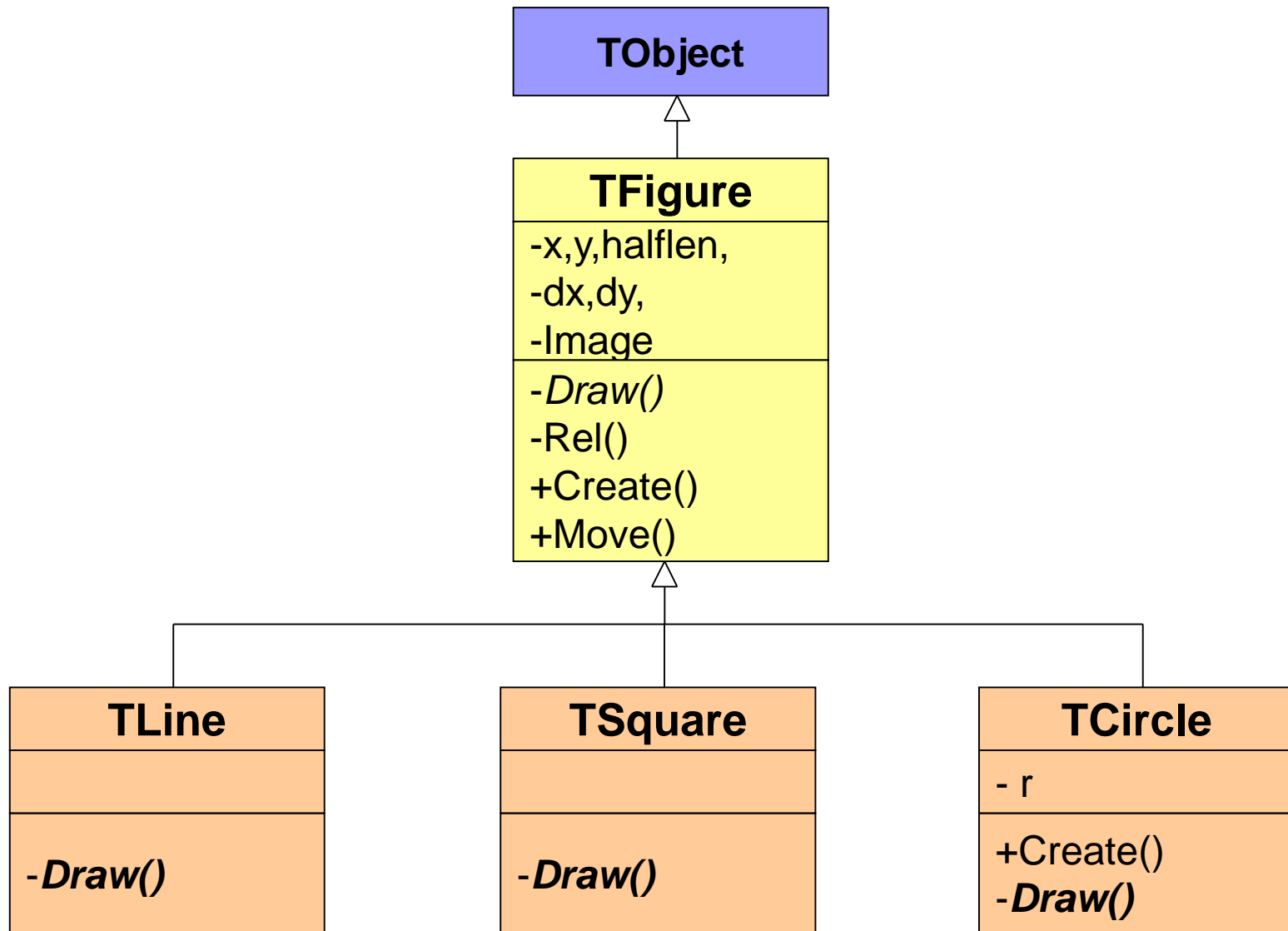
Объектная декомпозиция



$$dx = h \cdot \cos \alpha$$

$$dy = h \cdot \sin \alpha$$

Диаграмма классов предметной области



Модуль Figure

```
Unit Figure;
Interface Uses graphics,ExtCtrls;
Type TFigure=Class
    x,y, halflen,dx,dy:integer;          Image:TImage;
    constructor Create(ax,ay,ah:integer;aImage:TImage) ;
    procedure Move(t:single);
    procedure Draw;virtual;abstract;
    procedure Rel(t:real);
end;
TLine=Class(TFigure)
    procedure Draw;override;
end;
TSquare=Class(TFigure)
    procedure Draw;override;
end;
TCircle=Class(TFigure)
    r:integer;
    constructor Create(ax,ay,ah,ar:integer;
                      aImage:TImage) ;
    procedure DRAW;override;
end;
```

Модуль Figure (2)

Implementation

```
Constructor TFigure.Create;
```

```
Begin
```

```
    inherited Create;
```

```
    x:=ax; y:=ay; halflen:=ah; Image:=aImage;
```

```
End;
```

```
Procedure TFigure.Rel;
```

```
Begin
```

```
    dx:=round(halflen*cos(t));
```

```
    dy:=round(halflen*sin(t));
```

```
End;
```

```
Procedure TFigure.Move;
```

```
Begin
```

```
    Image.Canvas.Pen.Color:=clWhite;           Draw;
```

```
    Image.Canvas.Pen.Color:=clBlack;
```

```
    Rel(t);
```

```
    Draw;
```

```
End;
```

Модуль Figure (3)

```
Procedure TLine.Draw;
```

```
  Begin    Image.Canvas.MoveTo (x+dx, y+dy) ;  
          Image.Canvas.LineTo (x-dx, y-dy) ;
```

```
  End;
```

```
Procedure TSquare.Draw;
```

```
  Begin    Image.Canvas.MoveTo (x+dx, y+dy) ;  
          Image.Canvas.LineTo (x-dy, y+dx) ;  
          Image.Canvas.LineTo (x-dx, y-dy) ;  
          Image.Canvas.LineTo (x+dy, y-dx) ;  
          Image.Canvas.LineTo (x+dx, y+dy) ;
```

```
  End;
```

```
Constructor TCircle.Create;
```

```
  Begin    inherited Create (ax, ay, ah, aImage) ;    r:=ar;  
  End;
```

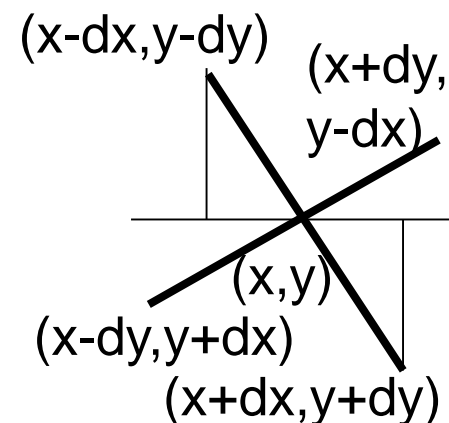
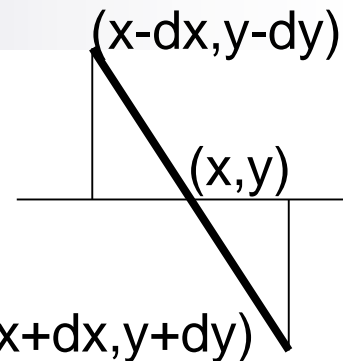
```
Procedure TCircle.Draw;
```

```
  Begin
```

```
    Image.Canvas.Ellipse (x+dx+r, y+dy+r, x+dx-r, y+dy-r) ;
```

```
  End;
```

```
end.
```



Описание класса окна

```
Unit Main;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes,
    Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
    TMainForm = class(TForm)
        Image: TImage;
        BeginButton, EndButton: TButton;
        Timer1: TTimer; // interval:=100
        procedure FormActivate(Sender: TObject);
        procedure BeginButtonClick(Sender: TObject);
        procedure EndButtonClick(Sender: TObject);
        procedure Timer1Timer(Sender: TObject);
    end;

Var MainForm: TMainForm;
```

Объявление объектов

```
implementation
```

```
uses Figure;
```

```
{ $R *.dfm }
```

```
Var
```

```
    t: single = 0.0;
```

```
    L: TLine;
```

```
    S: TSquare;
```

```
    C: TCircle;
```

```
procedure TMainForm.FormActivate(Sender: TObject);
```

```
begin
```

```
    Image.Canvas.Brush.Color := clWhite;
```

```
end;
```

Создание объектов и реализация движения

```
procedure TMainForm.BeginButtonClick(Sender: TObject) ;
begin
    L:=TLine.Create(60,100,50,Image) ;
    S:=TSquare.Create(180,100,50,Image) ;
    C:=TCircle.Create(300,100,50,10,Image) ;
    Timer1.Enabled:=true; // запуск таймера
end;
```

```
procedure TMainForm.Timer1Timer(Sender: TObject) ;
begin
    L.Move(t) ;
    S.Move(-0.2*t) ;
    C.Move(0.5*t) ;
    t:=t+0.5;
end;
```

Процедура организации движения

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

initialization

finalization

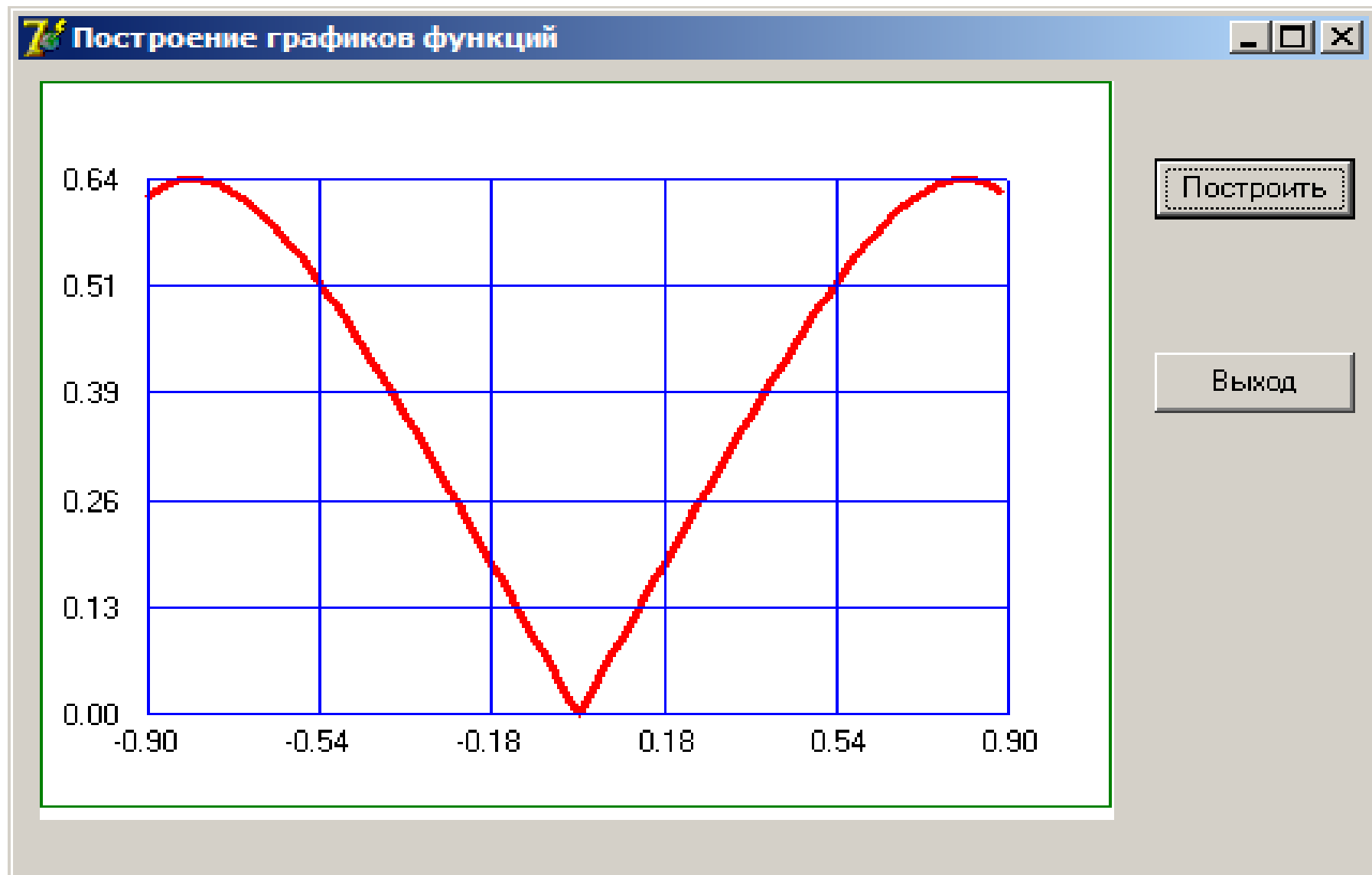
L.Free;

S.Free;

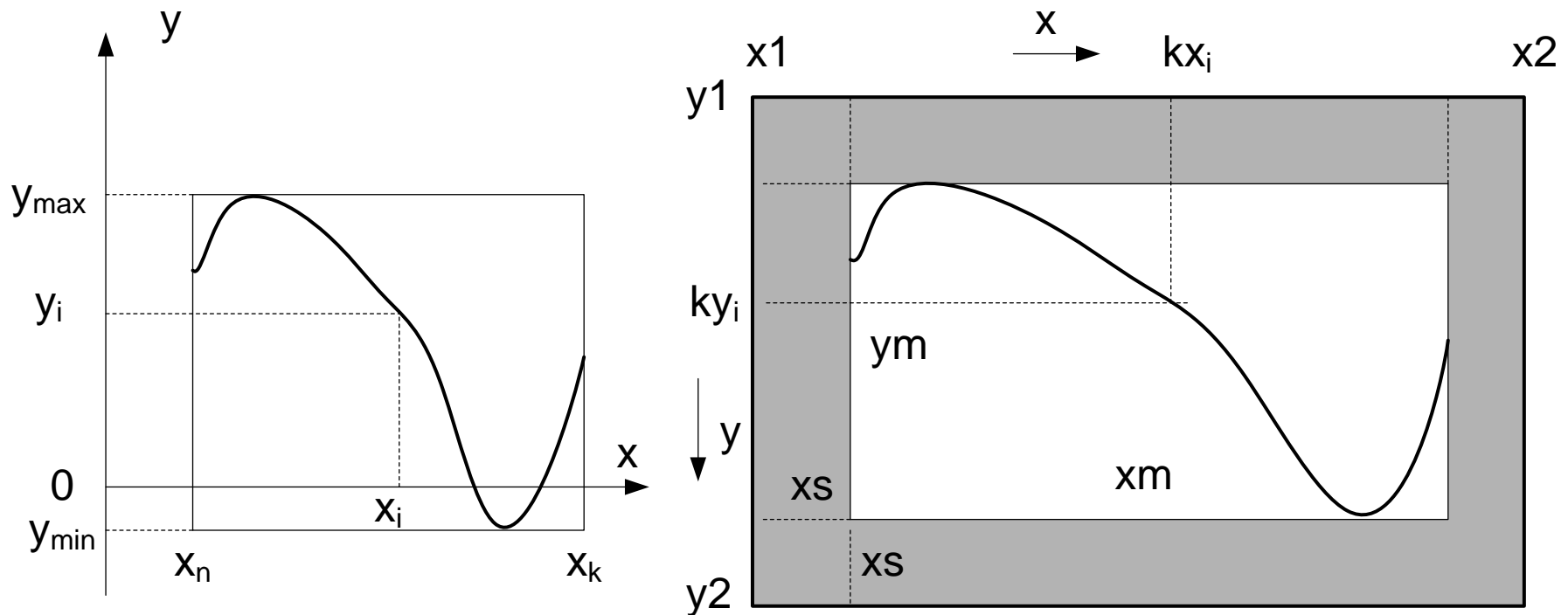
C.Free;

end.

Построение графика функции (Ex 8.4)



Отображение графика функции на экран



Масштабы по осям OX и OY:

$$mx = \frac{xm}{xk - xn}, \quad my = \frac{ym}{|ymax - ymin|}$$

Координаты точек в окне рисования:

$$kxi = \lfloor (xi - xn) \times mx \rfloor + x1 + xs,$$

$$kyi = \lfloor (ymax - yi) \times my \rfloor + y1 + xs.$$

Модуль Graphic

```
unit Graphic;
Interface
Uses ExtCtrls,Graphics;
Type  Fn=Function(X:single):single;

Procedure Run(xn, xk:single; n:integer; F:Fn;
              x1,y1,x2,y2:integer; Image:TImage);

Implementation
Procedure Run;
Var          arr:array[1..200] of record x,y:integer; end;
            x, y, dx, dx1, dyl, Fmin, Fmax, i,
            mx, my:single;      {масштабы}
            xm, ym,              {размер области окна рисования}
            xs, ys:integer;      {нижняя левая точка графика}
            s:string[10];
```

Модуль Graphic (2)

Begin

```
Image.Canvas.Pen.Color:=clGreen;  
Image.Canvas.Rectangle(x1,y1,x2,y2);    {рамка}  
xm:=x2-x1+1;  ym:=y2-y1+1;    {область графика}  
dx:=(xk-xn)/n;                    {шаг просчета точек}  
x:=xn;  y:=F(x);  
Fmin:=y; Fmax:=y;  
for i:=2 to n do    {табулируем функцию и }  
  begin            {вычисляем max и min}  
    x:=x+dx;        y:=F(x);  
    if y>Fmax then Fmax:=y;  
    if y<Fmin then Fmin:=y;  
  end;  
xs:=40;  ys:=ym-xs; {координаты нижней левой точки}  
mx:=(xm-xs*2)/(xk-xn);    {масштабы по x и y}  
my:=(ym-xs*2)/(Fmax-Fmin);
```


Модуль Graphic (3)

```
Image.Canvas.Pen.Color:=clRed;
Image.Canvas.Pen.Width:=3;
x:=xn;
for i:=1 to n do {считаем координаты точек}
begin
    arr[i].x:=round((x-xn)*mx)+x1+xs;
    arr[i].y:=round((Fmin-f(x))*my)+y1+ys;
    x:=x+dx;
end;
Image.Canvas.MoveTo(arr[1].x,arr[1].y);
for i:=2 to n do {соединяем точки}
    Image.Canvas.LineTo(arr[i].x,arr[i].y);
```

Модуль Graphic (4)

```
x:=xn;
dx1:=(xk-xn)/5;  {расстояние между линиями сетки}
Image.Canvas.Pen.Width:=1;
Image.Canvas.Pen.Color:=clBlue;
repeat           {вертикальные линии сетки и подпись}
    Str(x:5:2,s);
    Image.Canvas.TextOut(round((x-xn)*mx)+x1+xs-13,
                          ys+y1+5,s);
    Image.Canvas.MoveTo(round((x-xn)*mx)+x1+xs,
                          ym+y1-xs);
    Image.Canvas.LineTo(round((x-xn)*mx)+x1+xs,
                          y1+xs);

    x:=x+dx1;
until x>xk+0.00001;
```

Модуль Graphic (5)

```
y:=Fmin;  
dy1:=(fmax-fmin)/5;  
repeat {горизонтальные линии сетки и подпись}  
    Str(y:5:2,s);  
    Image.Canvas.TextOut(x1+5,  
        round(-(y-Fmin)*my)+ys+y1-6,s);  
    Image.Canvas.MoveTo(x1+xs,  
        round(-(y-Fmin)*my)+y1+ys);  
    Image.Canvas.LineTo(xm+x1-xs,  
        round(-(y-Fmin)*my)+y1+ys);  
    y:=y+dy1;  
until y>Fmax+0.00001;  
  
end;  
End.
```

Секция реализации модуля MainUnit

```
implementation
```

```
uses Graphic;
```

```
{ $R *.dfm }
```

```
Function f(x:single):single;
```

```
Begin    Result:=abs(0.64*sin(x*2));    End;
```

```
procedure TMainForm.FormActivate(Sender: TObject);
```

```
begin    Image.Canvas.Brush.Color:=clWhite; end;
```

```
procedure TMainForm.BeginButtonClick(Sender: TObject);
```

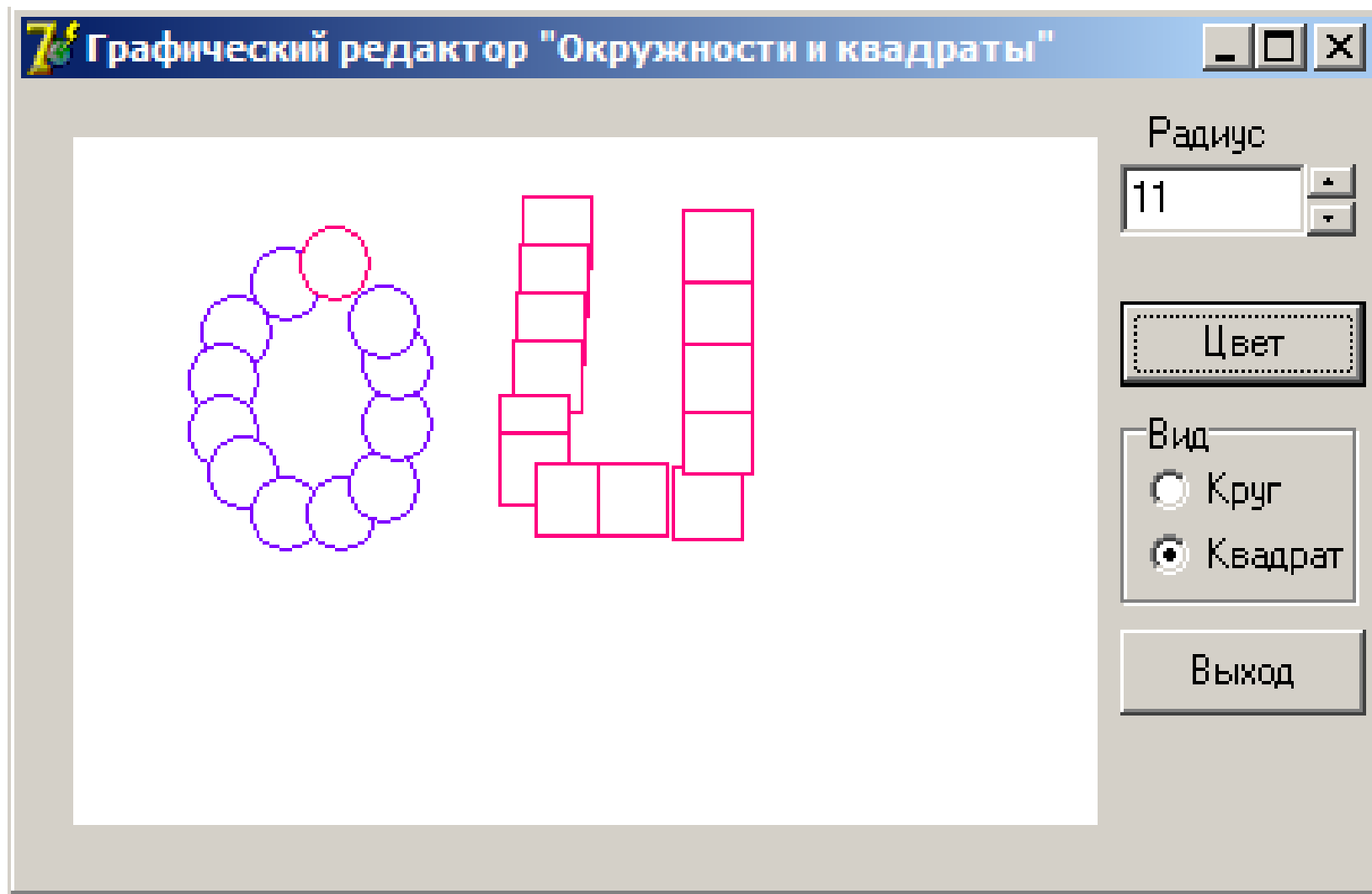
```
begin    Run(-0.9,0.9,100,f,0,0,400,300,Image);end;
```

```
procedure TMainForm.ExitButtonClick(Sender: TObject);
```

```
begin    Close;    end;
```

```
end.
```

Примитивный графический редактор (Ex 8.5)



Объектная декомпозиция

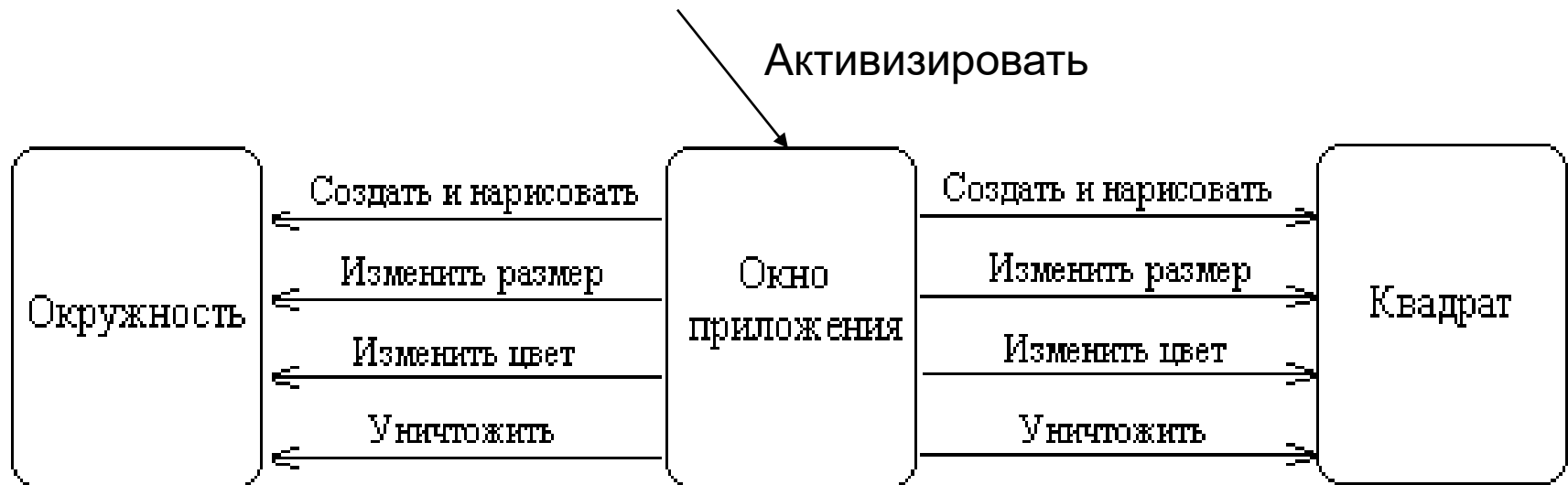
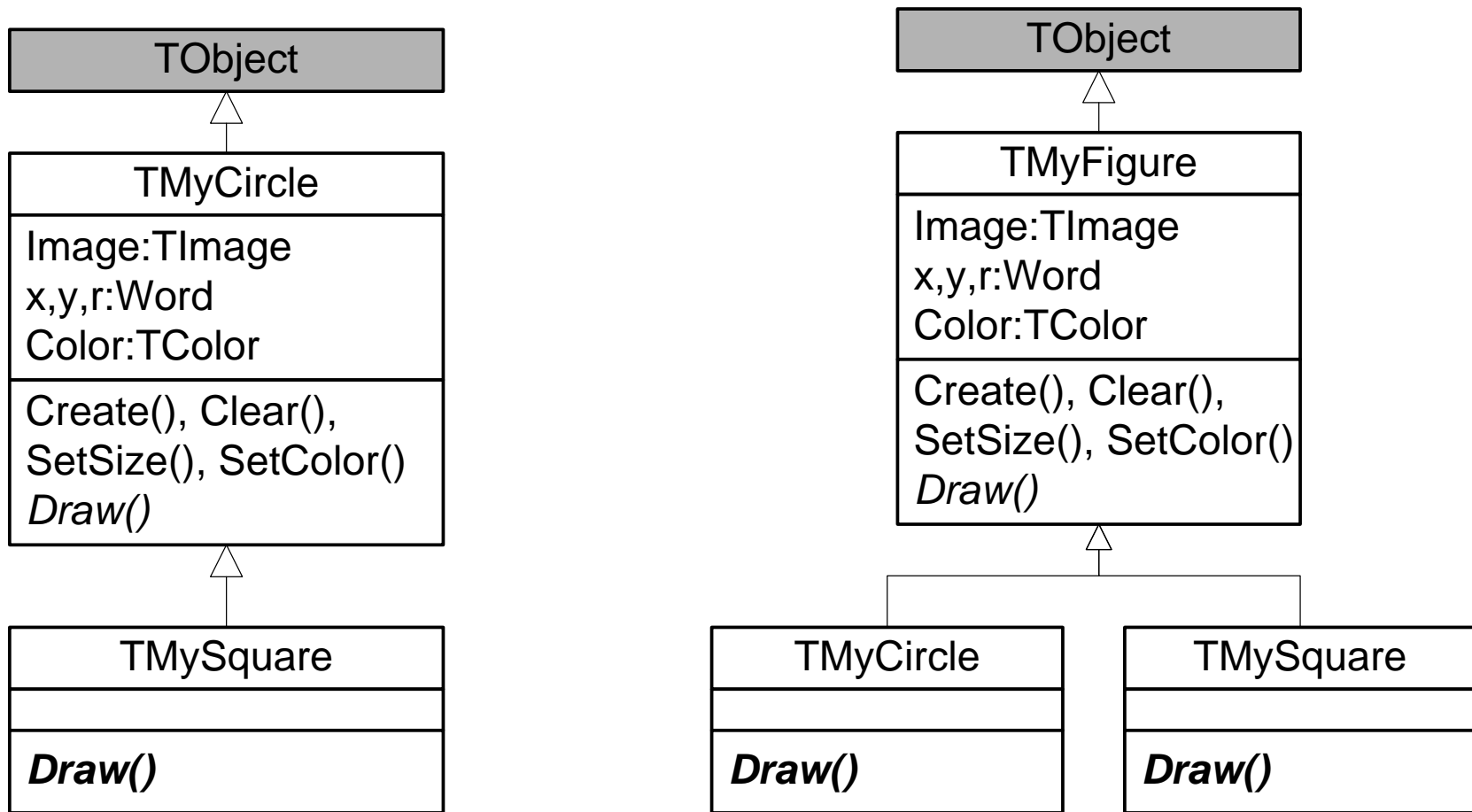


Диаграмма классов предметной области



Модуль Figure

```
Unit Figure;
```

```
Interface
```

```
Uses   extctrls,Graphics;
```

```
Type TMyFigure=class
```

```
    public
```

```
        x,y,Radius:Word;
```

```
        Color:TColor;
```

```
        Image:TImage;
```

```
        Procedure Draw; virtual; abstract;
```

```
        Procedure Clear;
```

```
        Constructor Create(aImage:TImage;
```

```
                           ax,ay,ar:Word;aColor:TColor) ;
```

```
    end;
```


Модуль Figure (2)

```
TMyCircle=class (TMyFigure)
    public    Procedure Draw; override;
end;
```

```
TMySquare=class (TMyFigure)
    public    Procedure Draw; override;
end;
```

Implementation

```
    Constructor TMyFigure.Create;
```

```
    Begin
```

```
        inherited Create;
```

```
        Image:=aImage;
```

```
        x:=ax;    y:=ay;
```

```
        Radius:=ar;
```

```
        Color:=aColor;
```

```
        Draw;
```

```
    End;
```

Модуль Figure (3)

```
Procedure TMyFigure.Clear;
```

```
  Var TempColor:TColor;
```

```
  Begin      TempColor:=Color;
```

```
            Color:=Image.Canvas.Brush.Color;
```

```
            Draw;
```

```
            Color:=TempColor;
```

```
  End;
```

```
Procedure TMyCircle.Draw;
```

```
  Begin
```

```
    Image.Canvas.Pen.Color:=Color;
```

```
    Image.Canvas.Ellipse (x-Radius,y-Radius,  
                          x+Radius,y+Radius) ;
```

```
  End;
```

Модуль Figure (4)

```
Procedure TMySquare.Draw;  
  Begin  
    Image.Canvas.Pen.Color:=Color;  
    Image.Canvas.Rectangle(x-Radius,y-Radius,  
                           x+Radius,y+Radius);  
  End;  
End.
```

Модуль Main

```
unit Main;  
  
interface  
  
uses  
    Windows, Messages, SysUtils, Variants, Classes, Graphics,  
    Controls, Forms, Dialogs, ComCtrls, StdCtrls, ExtCtrls;  
  
type  
    TMainForm = class(TForm)  
        Image: TImage;  
        ColorButton: TButton;  
        ExitButton: TButton;  
        RadioGroup: TRadioGroup;  
        rLabel: TLabel;  
        rEdit: TEdit;  
        UpDown: TUpDown;  
        ColorDialog: TColorDialog;
```

Модуль Main (2)

```
    procedure FormActivate(Sender: TObject);
    procedure ImageMouseDown(Sender: TObject;... );
    procedure UpDownClick(Sender: TObject; ...);
    procedure ColorButtonClick(Sender: TObject);
    procedure ExitButtonClick(Sender: TObject);
end;

var MainForm: TMainForm;
implementation
uses Figure;
Var C:TMyFigure;
{$R *.dfm}
procedure TMainForm.FormActivate(Sender: TObject);
begin
    Image.Canvas.Brush.Color:=clWhite;
    Image.Canvas.Pen.Color:=clBlack;
end;
```

Модуль Main (3)

```
procedure TMainForm.ImageMouseDown(Sender: TObject;  
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer) ;  
begin  
    if Button=mbLeft then  
        case RadioGroup.ItemIndex of  
            0: begin  
                C.Free;  
                C:=TMyCircle.Create(Image,X,Y,  
                    strtoint(rEdit.Text) , Image.Canvas.Pen.Color) ;  
            end;  
            1: begin  
                C.Free;  
                C:=TMySquare.Create(Image,X,Y,  
                    strtoint(rEdit.Text) , Image.Canvas.Pen.Color) ;  
            end;  
        end;  
    end;  
end;  
end;
```

Модуль Main (4)

```
procedure TMainForm.UpDownClick(Sender:TObject;  
                                Button:TUDBtnType) ;  
  
begin  
    if C<>nil then  
        begin C.Clear;  
              C.Radius:=strtoint(rEdit.Text) ; C.Draw;  
        end;  
    end;  
procedure TMainForm.ColorButtonClick(Sender: TObject) ;  
begin  
    if ColorDialog.Execute then  
        Image.Canvas.Pen.Color:=ColorDialog.Color;  
    if C<>nil then  
        begin C.Color:=Image.Canvas.Pen.Color;  
              Clear; Draw;          end;  
    end;  
end;
```

Модуль Main (5)

```
procedure TMainForm.ExitButtonClick(Sender:TObject) ;  
    begin    Close;    end;  
initialization  
finalization C.Free;  
  
end.
```