Составители Е.П.Храмцова

Редактор М.И.Антик

Методические указания посвящены изучению основ объектно-ориентированного программирования в среде создания приложений Delphi и моделирования на универсальном языке моделирования (UML). Они содержат: описание 5-й лабораторной работы, контрольные вопросы, учебную модель классов для 5, 6 и 7-й лабораторных работ, записанную на языке программирования Object Pascal. Разработка является продолжением методических указаний №№ 0426 и 0508. Материал предназначен для студентов, изучающих курс «Технологии программирования».

Печатаются по решению редакционно-издательского совета университета.

Рецензенты: В.Е.Красовский, А.М.Романов

Литературный редактор Н.К. Костыгина

Подписано в печать 10.04.2007. Формат 60x84 1/16.
Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,40. Усл.кр. - отт. 5,58. Уч.-изд.л.1,5.
Тираж 400 экз. Заказ 307. Бесплатно

Государственное образовательное учреждение
высшего профессионального образования
"Московский государственный институт радиотехники,
электроники и автоматики (технический университет)"
119454, Москва, пр. Вернадского, 78

ЛАБОРАТОРНАЯ РАБОТА №5

Цель работы: изучить

- особенности реализации унифицированного языка моделирования (UML) в CASE – инструментарии Model Maker for Delphi 7(8);
- правила записи диаграмм вариантов использования, классов и состояний в UML;
- основные элементы интегрированной среды разработчика Delphi как инструмента быстрой разработки приложений;
- модули, классы и их свойства в языке программирования Object Pascal;
- класс TForm, его основные свойства и обработчики событий в среде Delphi.

Постановка задачи

Заданы: файлы модулей UDiagram (Приложение 2) и ULabRab (Приложение 3) 99-го варианта в Object Pascal, модуля UForm1 1-й формы проекта Delphi 99-го варианта (Приложение 4). Исполняемая программа того же варианта – файл Lab5_V99.exe. Все перечисленные выше файлы находятся также в демонстрационной папке дисциплины ТП, расположенной на диске M:\Pascal\Demo\Lab5. Model Maker может вызываться из пункта главного меню Delphi 7.

Требуется

1. Исследовать предметную область (ПО) и сформулировать требования к создаваемому приложению. Построить концептуальную модель ПО в виде диаграммы вариантов использования.
2. Создать проект ПО:
   2.1. построить логическую модель структуры ПО в виде диаграммы классов;
   2.2. построить логическую модель поведения ПО в виде диаграммы состояний.
3. Осуществить программную реализацию проекта ПО:

3.1. создать проект в Delphi с именем Lab5_V<номер варианта 1-й лабораторной работы> с двумя формами: Form1 и Form2. Свойству Visible 2-й формы присвоить значение true;
3.2. поместить в Form1 3 объекта класса TButton, 2 объекта класса TTimer и объект класса TStaticText и придать этим объектам необходимые свойства;
3.3.присоединить к проекту модули UDiagram и ULabRab;
3.4. создать необходимые обработчики событий 3-х объектов класса TButton и 2-х объектов класса TTimer;
3.5. создать для Form1 обработчики событий OnActivate, OnDeactivate и OnPaint. Если активна Form2, то объекты классов модулей UDiagram и ULabRab неподвижны (см. Lab5_V99.exe);
3.6.изучить модули UDiagram и ULabRab и преобразовать их для своих фигур ПО.
4. Оформить отчёт. Отчёт должен содержать созданные концептуальную и логические модели ПО в виде диаграмм, и также ответы на вопросы.

Методические указания

Delphi – это популярная интегрированная среда разработчика приложений (Integrated Development Environment – IDE). Начиная с 7-й версии Delphi, в её состав включено CASE-средство визуального моделирования Model Maker. Языковой основой среды является язык Object Pascal, в котором введён новый тип class, являющийся последователем типа object в Turbo Pascal. Все классы Delphi порождены от абстрактного класса TObject. Этот класс не имеет полей и свойств, и содержит методы самого общего назначения, обеспечивающие жизненный цикл объектов – от их рождения до уничтожения. Среда Delphi предназначена для документирования, проектирования, кодирования, транслирования, выполнения и отлаживания приложений. Вид среды зависит от её настроек и действий программиста. Первоначально IDE состоит из следующих основных элементов: главного окна, инспектора объектов, конструктора формы и редактора кода. Главное окно состоит из главного меню, панели инструментов и палитры компонент. Первоначально конструктор формы изображается в виде пустого серого окна, на которое из палитры компонент визуально переносятся управляющие элементы (компоненты), такие, например, как метки, кнопки, таймеры и т.д. Инспектор объектов связан с конструктором формы и редактором кода. Инспектор объектов предназначен для изменения значений опубликованных (представленных в инспекторе объектов) свойств управляющего элемента, отмеченного на конструкторе формы. Инспектор объектов имеет две страницы: Properties и Events. Каждая страница инспектора объектов разделена на два столбца. Левый столбец содержит имена свойств, правый – их значения. Например, присвоение значения clRed свойству Color (страница Properties) формы обеспечивает её изображение в красном цвете.

Приложения Delphi управляются событиями. Событие может быть вызвано действием пользователя приложения (user events), логикой выполнения программы (internal events), а также взаимодействием программы с операционной системой (system events). Событие – это сигнал или сообщение, которое один объект посылает другому объекту или самому себе. Событие в Delphi – это свойство процедурного типа. Значением такого свойства является процедура, которая обрабатывает принятое объектом событие. Обработчики событий, перечисленных на странице Events инспектора объекта определены в классах-предках этого объекта. Однако реакцию на событие можно изменить, записав свой код в обработчике события. Для появления в редакторе 'пустого' (отсутствует код) обработчика события объекта произведите двойной щелчок мышью на значении события в инспекторе объекта и, далее, запишите в обработчике события программный код. Например, обработчик события OnResize (изменение размеров формы) изобразит форму приложения квадратной:

```
procedure TForm1.FormResize(Sender: TObject);
begin
Width:=Height
end;
```

В лабораторной работе создаются две формы: Form1 и Form2. На 1-й форме приложения появляются 7 объектов предыдущих

лабораторных работ (см. Lab5_V99.exe). Каждый из них может находиться в различных состояниях (виден, не виден, движется). Для запоминания состояния 1-й формы в момент её деактивизации и восстановления её состояния в момент её активизации необходимо записать программный код в обработчиках следующих событий: OnActivate, OnDeactivate, OnPaint. Программный код этих обработчиков представлен в Приложении 4.

В Delphi определены следующие пять типов свойств: простое, индексированное, массив, объект, интерфейс. Синтаксис простого свойства таков:

property <имя свойства>:<тип свойства>[**read**<имя поля или метода>] [**write**< имя поля или метода>] [**stored** <константа, имя поля или метода>] {[**default**< константа>] | [**nodefault**]};

где

read – спецификатор чтения значения свойства. Если при чтении свойства используется поле, то оно должно быть того же типа, что и тип свойства. Если при чтении свойства используется метод, то он должен быть подпрограммой-функцией без параметров, а её возвращаемое значение должно быть того же типа, что и тип свойства. Если спецификатор чтения свойства отсутствует, то свойство предназначено только для записи;

write – спецификатор записи значения свойства. Если при записи свойства используется поле, то оно должно быть того же типа, что и тип свойства. Если при записи свойства используется метод, то он должен быть подпрограммой-процедурой с одним параметром-значением или параметром-константой того же типа, что и тип свойства. Этот параметр задаёт значение, которое помещается в свойство. Если спецификатор записи свойства отсутствует, то свойство предназначено только для чтения;

stored – спецификатор сохранения значения свойства. Константа, поле или функция без параметров, записываемая после спецификатора, должны быть логического типа. Если константа, значение поля или функции равно true, то значение свойства запоминается в файле формы (*.dfm), если false – то не запоминается;

default – спецификатор предназначен для запоминания значения свойства класса в файле формы. Константа, записываемая

после спецификатора, может быть только простого типа, за исключением вещественного и строкового типа. Свойству объекта класса присваивается значение или в конструкторе класса или по умолчанию - при создании объекта класса. Если значение свойства совпадает со значением, указанным в рассматриваемом спецификаторе, то оно не записывается в файле формы, если не совпадает, то запоминается;

nodefault – спецификатор, отменяющий значение по умолчанию свойства, унаследованного от класса-предка.

При объявлении свойства нельзя опускать одновременно спецификаторы **read** и **write**. Спецификатор **stored** отменяет умалчиваемое значение и поэтому он не может быть записан вместе со спецификатором **default**.

Например, простое свойство Visible в классе TFigure может быть записано так:

property Visible:boolean **read** fVisible **write** SetVisible;

где fVisible – скрытое (private) поле логического типа;

SetVisible – скрытая процедура класса, имеющая вид:

```
procedure TFigure.SetVisible(aVisible:boolean);
 begin
  fVisible:=aVisible;
  if fVisible then
     begin
      if fColor=BackColor
         then  fColor:=fColorBegin;
      draw
     end
        else done
 end;
```

Свойства используемых в данной лабораторной работе компонент приведены в Приложении 1.

Вопросы

Дайте определения и/или приведите примеры

1. Сложной системы. Основных принципов объектно-ориентированного моделирования сложных систем.

2. События, обработчика события.

Ответьте на вопросы и приведите примеры

3. Для чего предназначена диаграмма вариантов использования? На каком этапе создания программной модели системы она строится? Что такое актёр и роль? Какие отношения между вариантами использования определены в UML?
4. Для чего предназначены диаграммы классов? Как изображается класс? Каков формат записи его атрибутов и методов в языке UML? Какие основные отношения между классами определены в UML? Как обозначаются эти отношения? Какими отношениями связаны классы TForm1 и TButton, TForm1 и TMyFigure1, TFigure и TMyFigute2 лабораторной работы?
5. Чем отличается изображение интерфейса на диаграмме вариантов использования от его изображения на диаграмме классов?
6. На каком этапе создания программной модели системы создаётся диаграмма состояний? Для каких элементов моделируемой системы строится диаграмма состояний? Сколько состояний имеет объект Form2? Когда Form2 переходит из одного состояния в другое? Как изображаются начальное и конечное состояния этого объекта?
7. Чем отличается деятельность от действия? Каков формат записи деятельности и действия и где они записываются на диаграмме состояния?
8. Как на диаграмме состояний обозначаются параллельные переходы? Что такое подсостояние? Приведите примеры параллельных и последовательных подсостояний.
9. Какие псевдосостояния определены в UML и как они обозначаются? Для чего предназначено историческое состояние, каковы его разновидности и как они обозначаются?
10. Чем отличаются классы и модули языков программирования Turbo Pascal и Object Pascal?
11. Чем форма Delphi отличается от других компонент? Какой тип имеют объекты Form1 и Form2? Каков их общий ближайший класс-предок?
12. Существует ли связь между заголовком формы и именем модуля формы? Какое свойство формы соответствует её заголовку?

13. Как влияет удаление компонента из конструкора формы на редактор формы? Как влияет добавление компонента в конструктор формы на редактор формы?
14. Какие типы свойств определены в Delphi? Каков синтаксис простого свойства? Может ли одно свойство влиять на другое? Что обеспечивает свойство Hint и как оно связано со свойством ShowHint объектов Form1 и Form2?
15. Является ли обработчик события свойством? Как добавить обработчик события в редактор формы? Как удалить обработчик из редактора формы? Обработчик какого события создаётся в редакторе формы по умолчанию (при щелчке мышью на форме)?
16. Какие обработчики событий (свойства) были созданы в лабораторной работе для объекта Form1? Для чего предназначено свойство формы OnPaint? Как изменится выполнение приложения, если значение этого свойства удалить из проекта?
17. На какое событие реагирует невизуальный компонент класса TTimer? Когда это событие происходит?

ПРИЛОЖЕНИЕ 1

Свойства объектов лабораторной работы

1. Визуальные компоненты (опубликованные свойства)

*1.1 классов **TForm1** и **TForm2***

Свойство	Страница инспектора объектов	Назначение
BorderIcons	Properties	Предоставляет возможность свёртывания, развёртывания и удаления формы в режиме выполнения приложения
Caption	Properties	Содержит название формы
Constraints	Properties	Определяет максимальные и минимальные размеры формы

Enabled	Properties	Указывает, реагирует ли форма на события мыши, клавиатуры и системных таймеров
Position	Properties	Позиционирует форму на экране
Visible	Properties	Определяет видимость формы в режиме выполнения приложения
OnPaint	Events	Выполняет программный код метода при событии перерисовки формы
OnActivate	Events	Выполняет программный код метода при событии активизации формы
OnCreate	Events	Выполняет программный код метода при конструировании формы
OnDeactivate	Events	Выполняет программный код метода при событии деактивизации формы

*1.2 класса **TButton**(закладка Standard палитры компонент)*

Свойство	Страница инспектора объектов	Назначение
Caption	Properties	Содержит название кнопки
Enabled	Properties	Указывает, реагирует ли кнопка на события мыши, клавиатуры и системных таймеров
Visible	Properties	Определяет видимость кнопки в режиме выполнения приложения
OnClick	Events	Выполняет программный код метода при щелчке левой кнопкой мыши на объекте

*1.3 класса **TStaticText** (закладка Additional палитры компонент, страница Properties инспектора объектов)*

Свойство	Назначение
Alignment	Определяет способ выравнивания текста внутри метки
BevelKind	Определяет стиль внешней и внутренней рамок метки
Caption	Определяет строку метки, выводимую на компонент-владелец метки
Height	Определяет высоту текстовой метки
Width	Определяет ширину текстовой метки

2. Невизуальный компонент (опубликованные свойства)

*Класс **TTimer** (закладка System палитры компонент)*

Свойство	Страница инспектора объектов	Назначение
Enabled	Properties	Определяет доступность таймера (по умолчанию = true)
Interval	Properties	Указывает интервал посылки таймером периодически происходящего события (по умолчанию = 1секунде (1000 миллисекунд))
OnTimer	Events	Определяет метод, который будет выполнен если таймер доступен, с интервалом, определённом в свойстве таймера Interval

3. Объекты классов модулей UDiagram и ULabRab

Классы ***TPoint, TCircle, TLine, TMyFigure1, TMyFigure2, TMyFigure3*** имеют свойство Visible, определяющее видимость фигуры в приложении.

ПРИЛОЖЕНИЕ 2

Модуль UDiagram в Delphi

```
unit UDiagram;
interface
 uses  Graphics, Forms, Dialogs;
 type
// Класс TFigure – родоначальник классов геометр. фигур.
    TFigure =class(TObject)
     private
      fVisible:boolean; //Видимость объекта
      fColor:TColor; //Цвет объекта
      fColorBegin:TColor; //Начальный цвет объекта
      BackColor:TColor; //Цвет фона
      x,y,invertX,invertY:integer;
      u:real;
      procedure SetVisible(aVisible:boolean);
     public
      procedure move(dx,dy:integer);
      procedure moveTraject(dx:integer);
      constructor Create(ax,ay:integer; aColor:TColor;
                         Sender:TObject);
      procedure done;                    virtual;
      procedure dx_dy(dx,dy:integer);    virtual;
      procedure draw;                    virtual;
     protected
      MotherForm:TForm;
     published
     // property Color: TColor - это свойство необходимо для
     //выполнения сложных вариантов 6-й лабораторной работы
     //(по примерам файлов N5,6,7)
      property Visible:boolean read fVisible write SetVisible;
    end;
    TPoint=class(TFigure)
     public
      procedure done;                override;
```



```
      procedure draw;                override;
    end;
    TCircle=class(TPoint)
     private
      r:integer;
     public
      constructor Create(axc,ayc,ar:integer; acolor_c:TColor;
                                    Sender:TObject);
      procedure done;                override;
      procedure draw;                override;
    end;
    TLine=class(TFigure)
     private
      x1,y1,x2,y2:integer;
     public
      constructor Create(ax1,ay1,ax2,ay2:integer;  acolor_l:TColor;
                         Sender:TObject);
      procedure done;                        override;
      procedure dx_dy(dxln,dyln:integer);override;
      procedure draw;                        override;
    end;
 implementation
// Методы и свойство класса TFigure
constructor TFigure.Create(ax,ay:integer;
                           aColor:TColor;Sender:TObject);
    begin
    try
     MotherForm:=Sender as TForm;
     if ax<0 then
          raise EAbort.Create('Увеличьте значение пара-
                              метра "ax" конструктора объекта');
     if ax>MotherForm.ClientWidth then
          raise EAbort.Create('Уменьшите значение пара-
                              метра "ax" конструктора объекта');
     if ay<0 then
          raise EAbort.Create('Увеличьте значение пара-
```

```
                    метра "ay" конструктора объекта');
    if ay>MotherForm.ClientHeight then
            raise EAbort.Create('Уменьшите значение пара-
                    метра "ay" конструктора объекта');
    BackColor:= MotherForm.Color;
    if aColor <> BackColor
     then
      begin
       fColor:=aColor;
       fColorBegin:=aColor
      end
     else
      begin
       fColor:=clGreen;
       fColorBegin:=clGreen
      end;
     invertX:=1;
     invertY:=1;
     u:=0.0;
     x:=ax;
     y:=ay
    except
      on E:Eabort do ShowMessage(E.Message)
    end
   end;
procedure TFigure.SetVisible(aVisible:boolean);
   begin
    fVisible:=aVisible;
    if fVisible then
        begin
         if fColor=BackColor then  fColor:=fColorBegin;
         draw
        end
              else done
   end;
procedure TFigure.move(dx,dy:integer);
```



```
   begin
    done;
    dx_dy(dx,dy);
    draw
   end;
procedure TFigure.moveTraject(dx:integer);
   const du=Pi/120;
   var dy:integer;
   begin
    dy:=round(4*sin(u)); // Обращение к функции
    move(invertX*dx, invertY*dy);
    u:=u+du;
    if (x>=MotherForm.ClientWidth) or (x<=0)
                          then invertX:=-invertX;
    if (y<= 0) or (y>=MotherForm.ClientHeight)
                          then invertY:=-invertY
   end;
  procedure TFigure.done;
   begin
    MotherForm.Canvas.Pen.Color:=BackColor;
    fVisible:=false
   end;
  procedure TFigure.draw;
   begin
    MotherForm.Canvas.Pen.Color:=fColor;
    fVisible:=fColor<>BackColor
   end;
  procedure TFigure.dx_dy(dx,dy:integer);
   begin
    x:=x+dx;
    y:=y+dy
   end;
// Методы класса TPoint
   procedure TPoint.done;
    begin
     inherited done;
```

```
    MotherForm.Canvas.Ellipse(x,y,x+3,y+3)
   end;
  procedure TPoint.draw;
   begin
    inherited draw;
    MotherForm.Canvas.Ellipse(x,y,x+3,y+3)
   end;
// Методы класса TCircle
   constructor TCircle.Create(axc,ayc,ar:integer; acolor_c:TColor;
                              Sender:TObject);
    begin
     inherited Create(axc,ayc,acolor_c,Sender);
     r:=ar
    end;
   procedure TCircle.done;
    begin
     MotherForm.Canvas.Pen.Color:=BackColor;
     MotherForm.Canvas.Ellipse(x-r,y,x+r,y+2*r);
     fVisible:=false
    end;
   procedure TCircle.draw;
    begin
     MotherForm.Canvas.Pen.Color:=fColor;
     MotherForm.Canvas.Ellipse(x-r,y,x+r,y+2*r);
     fVisible:=fColor<>BackColor
    end;
// Методы класса TLine
   constructor TLine.Create(ax1,ay1,ax2,ay2:integer;
                            acolor_l:TColor; Sender:TObject);
    begin
     inherited Create((ax1+ax2) div 2, (ay1+ay2) div 2, acolor_l,
                                                        Sender);
     x1:=ax1;
     y1:=ay1;
     x2:=ax2;
     y2:=ay2
```



```
    end;
   procedure TLine.done;
    begin
     inherited done;
     MotherForm.Canvas.MoveTo(x1,y1);
     MotherForm.Canvas.LineTo(x2,y2)
    end;
   procedure TLine.dx_dy(dxln,dyln:integer);
    begin
     inherited dx_dy(dxln,dyln);
     x1:=x1+dxln;
     y1:=y1+dyln;
     x2:=x2+dxln;
     y2:=y2+dyln
    end;
   procedure TLine.draw;
    begin
     inherited draw;
     MotherForm.Canvas.MoveTo(x1,y1);
     MotherForm.Canvas.LineTo(x2,y2)
    end;
end.
```

ПРИЛОЖЕНИЕ 3

Модуль ULabRab в Delphi

```
unit ULabRab;
interface
uses Graphics, UDiagram;
type
// Класс 1-й лабораторной работы
TMyFigure1=class(TLine)
   private
    x3,y3,x4,y4 : integer;
   public
```

```
    constructor Create(xfig,yfig,L:integer; acolor_fig:TColor;
                                              Sender: TObject);
    procedure dx_dy(dxfig,dyfig:integer);override;
    procedure draw;                override;
    procedure done;                override;
 end;
// Класс 2-й лабораторной работы
 TMyFigure2=class(TFigure)
   private
    Circle:TCircle; Figure1:TMyFigure1;
   public
    constructor Create(xfig,yfig,L:integer; acolor_fig:TColor;
                                              Sender: TObject);
    procedure dx_dy(dxfig,dyfig:integer);override;
    procedure draw;                override;
    procedure done;                override;
 end;
// Класс 3-й лабораторной работы
 TMyFigure3=class(TFigure)
   private
    figures: array[1..6] of  TFigure;
   public
     constructor Create(xfig,yfig,L:integer; acolor_fig:TColor;
                                              Sender: TObject);
    procedure dx_dy(dxfig,dyfig:integer);override;
    procedure draw;                override;
    procedure done;                override;
 end;
implementation
// Методы класса TMyFigure1
    constructor TMyFigure1.Create(xfig,yfig,L:integer;
                        acolor_fig:Tcolor; Sender: TObject);
    begin
     inherited Create(xfig - L div 2, yfig, xfig+ L div 2, yfig,
                                          acolor_fig, Sender);
     x3:=xfig;
```



```
     y3:=yfig - L div 2;
     x4:=xfig;
     y4:=yfig + L div 2;
    end;
   procedure TMyFigure1.done;
    begin
     inherited done;
     MotherForm.Canvas.MoveTo(x3,y3);
     MotherForm.Canvas.LineTo(x4,y4);
    end;
   procedure TMyFigure1.dx_dy(dxfig,dyfig:integer);
    begin
     inherited dx_dy(dxfig,dyfig);
     x3:=x3+dxfig;
     y3:=y3+dyfig;
     x4:=x4+dxfig;
     y4:=y4+dyfig;
    end;
   procedure TMyFigure1.draw;
    begin
     inherited draw;
     MotherForm.Canvas.MoveTo(x3,y3);
     MotherForm.Canvas.LineTo(x4,y4);
    end;
// Методы класса TMyFigure2
    constructor TMyFigure2.Create(xfig,yfig,L:integer;
                        acolor_fig:Tcolor; Sender: TObject);
    begin
     inherited Create(xfig,yfig,acolor_fig, Sender);
     Circle:=TCircle.Create(xfig, yfig-L div 2, L div 2, acolor_fig,
                                          Sender);
     Figure1:=TMyFigure1.Create(xfig, yfig, L, acolor_fig, Sender)
    end;
   procedure TMyFigure2.dx_dy(dxfig, dyfig:integer);
    begin
     inherited dx_dy(dxfig,dyfig);
```

```
  Circle.dx_dy(dxfig,dyfig);
  Figure1.dx_dy(dxfig,dyfig)
 end;
procedure TMyFigure2.draw;
 begin
  inherited draw;
  Circle.draw;
  Figure1.draw
 end;
procedure TMyFigure2.done;
 begin
  inherited done;
  Circle.done;
  Figure1.done
 end;
// Методы класса TMyFigure3
 constructor TMyFigure3.Create(xfig,yfig,L:integer;
                          acolor_fig:Tcolor; Sender: TObject);
 begin
  inherited Create(xfig,yfig,acolor_fig, Sender);
  figures[1]:=TMyFigure2.Create(xfig-L*2, yfig, L, acolor_fig, .
                Sender);
  figures[2]:=TCircle.Create(xfig-L,yfig-L div 2, L div 2,
                                        acolor_fig, Sender);
  figures[3]:=TMyFigure1.Create(xfig,yfig,L,acolor_fig, Sender);
  figures[4]:=TMyFigure2.Create(xfig+L,yfig,L,acolor_fig,
                                                Sender);
  figures[5]:=TCircle.Create(xfig+L*2,yfig-L div 2,L div 2,
                                        acolor_fig, Sender);
  figures[6]:=TLine.Create(xfig+L*2,yfig-L div 2,
                xfig+L*2,yfig+L div 2,acolor_fig, Sender);
 end;
procedure TMyFigure3.dx_dy(dxfig,dyfig:integer);
 var i:byte;
 begin
  inherited dx_dy(dxfig,dyfig);
```



```
  for i:=1 to Length(Figures) do
   figures[i].dx_dy(dxfig,dyfig)
 end;
procedure TMyFigure3.draw;
 var i:byte;
 begin
  inherited draw;
  for i:=1 to Length(Figures) do
   figures[i].draw
 end;
procedure TMyFigure3.done;
 var i:byte;
 begin
  inherited done;
  for i:=1 to Length(Figures) do
   figures[i].done
 end;
end.
```

ПРИЛОЖЕНИЕ 4

Модуль UForm1 в Delphi

```
unit UForm1;
interface
uses Windows, Messages, SysUtils,Variants, Classes, Graphics,
 Controls, Forms, Dialogs, ExtCtrls, StdCtrls;
type
  TForm1 = class(TForm)
   Button1: TButton;
   Button2: TButton;
   Button3: TButton;
   Timer1: TTimer;
   Timer2: TTimer;
   StaticText1: TStaticText;
   procedure Timer1Timer(Sender: TObject);
   procedure Timer2Timer(Sender: TObject);
```

```
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormDeactivate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
end;
var Form1: TForm1;
Implementation
  uses UDiagram, ULabRab;
  var pn:TPoint; cl:TCircle; ln:TLine; myf1:TMyFigure1;
     myf2:TMyFigure2; myf3,myf4:TMyFigure3;
     t1,t2:boolean;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  pn.move(4,-3);
  cl.move(2,2);
  ln.move(1,2);
  myf1.move(2,0);
  myf2.move(0,1);
  myf3.move(3,1)
end;
procedure TForm1.Timer2Timer(Sender: TObject);
begin
  myf4.moveTraject(2)
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  Constraints.MinHeight:=500;
  Constraints.MinWidth:=800;
  Constraints.MaxHeight:=500;
  Constraints.MaxWidth:=800;
  Position:=poScreenCenter;
  Timer1.Enabled:=False;
  Timer2.Enabled:=False;
```



```
  Timer1.Interval:=6;
  Timer2.Interval:=2;
  Button3.Visible:=false;
  Pn:=TPoint.Create(120,370,clAqua,Sender);
  Cl:=TCircle.Create(10,30,30,clBlue,Sender);
  Ln:=TLine.Create(60,200,260,300,clRed,Sender);
  MyF1:=TMyFigure1.Create(70,250,50,clYellow,Sender);
  MyF2:=TMyFigure2.Create(230,150,50,clGreen,Sender);
  MyF3:=TMyFigure3.Create(170,120,50,clBlack,Sender);
  MyF4:=TMyFigure3.Create(200,60,35,clWhite,Sender)
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  Pn.Visible:=True;
  Cl.Visible:=True;
  Ln.Visible:=True;
  Button1.Visible:=false;
  if not Button2.Visible then Button3.Visible:=true
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  MyF2.Visible:=True;
  MyF1.Visible:=True;
  MyF3.Visible:=True;
  MyF4.Visible:=True;
  Button2.Visible:=false;
  if not Button1.Visible then Button3.Visible:=true
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  timer1.Enabled:=true;
  timer2.Enabled:=true;
  Button3.Visible:=false;
  t1:=true;
  t2:=true
end;
```

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  timer1.Enabled:=t1;
  timer2.Enabled:=t2
end;
procedure TForm1.FormDeactivate(Sender: TObject);
begin
  t1:=Timer1.Enabled;
  t2:=Timer1.Enabled;
  timer1.Enabled:=false;
  timer2.Enabled:=false
end;
procedure TForm1.FormPaint(Sender: TObject);
begin
  if pn.Visible then
     begin
       pn.Visible:=true;
       cl.Visible:=true;
       ln.Visible:=true
     end;
  if myf1.Visible then
     begin
       myf1.Visible:=true;
       myf2.Visible:=true;
       myf3.Visible:=true;
       myf4.Visible:=true
     end
end;
end.
```