МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Юго-Западный государственный университет

(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«14» 04 2022 г.

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания по выполнению практических работ

для студентов направления подготовки 09.03.01

Курск 2022

УДК 621.3

Составитель: Э.И. Ватутин

Рецензент
Кандидат технических наук, доцент *Т.Н. Конаныхина*

Объектно-ориентированное программирование: методические указания по выполнению практических работ по дисциплинам «Программирование», «Объектно-ориентированное программирование» / Юго-Зап. гос. ун-т; сост.: Э.И. Ватутин; Курск, 2022. 17 с.: ил. 4.

Методические рекомендации содержат сведения по разработке программ с использованием объектно-ориентированной технологии на современных языках программирования высокого уровня.

Предназначены для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60x84 1/16.
Усл. печ. л. Уч. – изд.л. Тираж 30 экз. Заказ 314. Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул. 50 лет Октября, 94.

Содержание

Основные понятия по работе с 2d-графикой в Delphi

Для работы с графикой в составе визуальных компонентов Delphi присутствует свойство `Canvas`, инкапсулирующее в своем составе весь необходимый функционал (карандаши, кисти, подпрограммы по рисованию графических примитивов). Перерисовку компонента необходимо производить не в произвольный момент времени, а при получении сообщения `WM_PAINT`, отправляемого ОС Windows. Для его обработки у визуальных компонентов VCL предусмотрен обработчик событий `OnPaint`, в тело которого необходимо вставить необходимый код, связанный с отрисовкой компонента. Рисование можно осуществлять на любом визуальном компоненте, однако рекомендуется выполнять его с использованием компонента VCL PaintBox. Простейший пример рисования приведен ниже:

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with PaintBox1.Canvas do begin
    Pen.Color := clRed;
    MoveTo(0, 0);
    LineTo(100, 100);

    Brush.Color := clGreen;
    Rectangle(30, 30, 70, 70);
  end;
end;
```

В приведенном примере с использованием карандаша (Pen) красного цвета сперва производится рисование линии, а затем – прямоугольника, заливка которого делается кистью (Brush) зеленого цвета. Для карандаша и кисти кроме цвета можно задавать различные параметры (например, тип штриховки).

Если в какой-либо момент времени необходимо перерисовать некоторый визуальный компонент, не дожидаясь прихода сообщения `WM_PAINT`, необходимо вызвать метод `Invalidate()`, что приведет к

активации обработчика `OnPaint` с последующим обновлением графического представления компонента на форме.

В случае, если перерисовка компонента занимает много времени, бывает целесообразно рисование на т.н. виртуальном экране, содержимое которого впоследствии отрисовывается на визуальном компоненте. Для этого применяется класс `TBitmap`. При рисовании иногда наблюдается неприятное моргание, чего можно избежать путем задания соответствующего свойства компонента

```
ControlStyle := ControlStyle + [csOpaque];
```

Соответствующий пример приведен ниже.

```
var
  Bmp: TBitmap;


procedure TForm1.FormCreate(Sender: TObject);
begin
  Bmp := TBitmap.Create();
  Bmp.Width := PaintBox1.ClientWidth;
  Bmp.Height := PaintBox1.ClientHeight;

  Form1.ControlStyle := Form1.ControlStyle + [csOpaque];
end;


procedure TForm1.Button1Click(Sender: TObject);
begin
  with Bmp.Canvas do begin
    Действия по рисованию
  end;
  PaintBox1.Invalidate();
end;


procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0, 0, Bmp);
end;
```

Замечание. Рисование через компоненты VCL является удобным, но не очень эффективным в плане быстродействия. Существенно более эффективным способом является работа с Device Context (DC) компонентов напрямую через вызовы WinAPI-функций.

Описание предметной области

В рассмотренном ниже примере производится рисование предметов, летающих в пределах клиентской области PaintBox'a и отражающихся от его краев. В качестве предметов будем рассматривать материальную точку с координатами $x = (x, y)$ и скоростью движения на плоскости $\vec{V} = (v_x, v_y)$ и окружность радиуса *R*. Окружность и материальная точка имеют заданный цвет *C*. В качестве вариантов движения реализуем движение по прямой и по спирали, для чего потребуется задание радиуса вращения *r* значений текущего угла a и угловой скорости w. При движении по спирали координаты центра окружности определяются как

$$x = x_{тек} + r\cos a,$$
$$y = y_{тек} + r\sin a.$$

Движение графических примитивов будем реализовывать по шагам. Один шаг движения по прямой описывается следующими изменениями значений координат:

$$x_{тек} = x_{пред} + v_x,$$
$$y_{тек} = y_{пред} + v_y,$$

а один шаг движения по спирали – изменением значения угла:

$$a_{тек} = a_{пред} + w.$$

При попадании центра графического примитива за пределы области рисования соответствующий вектор скорости меняет свой знак.

Разработка объектно-ориентированной иерархии классов

Функционал материальной точки (координаты, скорости, цвет) инкапсулирован в составе класса `TFly`. Кроме соответствующий полей он включает в своем составе виртуальный конструктор `Create`, производящий инициализацию полей псевдослучайными значениями, виртуальный метод

Draw для отрисовки текущего графического примитива на заданном Bitmap'е и виртуальный метод DoStep для совершения одной итерации движения. Класс TFly является базовым классом разрабатываемой объектно-ориентированной иерархии классов. Для создания объектов от заранее неопределенных классов далее будем использовать метакласс TFlyClass. Соответствующий программный код класса и объявление метакласса приведены ниже.

```
unit Main;

interface

uses
  Windows, Graphics;

const
  MAX_V = 10;

var
  MAX_X, MAX_Y: Integer;

type
  TFlyClass = class of TFly;

  TFly = class
  protected
    fX, fY, fVx, fVy: Integer;
    fColor: TColor;
  public
    property X: Integer read fX write fX;
    property Y: Integer read fY write fY;
    property Vx: Integer read fVx write fVx;
    property Vy: Integer read fVy write fVy;
    property Color: TColor read fColor write fColor;

    constructor Create(); virtual;
    procedure Draw(Bmp: TBitmap); virtual;
    procedure DoStep(); virtual;
  end;

...


implementation

{ TFly }

constructor TFly.Create();
begin
  fX := Random(MAX_X);
  fY := Random(MAX_Y);

  fVx := Random(MAX_V*2) - MAX_V;
  fVy := Random(MAX_V*2) - MAX_V;
```

```
  fColor := RGB(Random(256), Random(256), Random(256));
end;


procedure TFly.Draw(Bmp: TBitmap);
begin
  with Bmp.Canvas do begin
    Pen.Color := fColor;
    Brush.Color := fColor;
    Ellipse(fX-2, fY-2, fX+2, fY+2);
  end;
end;


procedure TFly.DoStep();
begin
  fX := fX + fVx;
  if (fX < 0) or (fX > Manager.fBmp.Width) then
    fVx := -fVx;

  fY := fY + fVy;
  if (fY < 0) or (fY > Manager.fBmp.Height) then
    fVy := -fVy;
end;

...
```

Для рисования графических примитивов разработан класс `TDrawManager`. Он включает в своем составе поля для хранения Bitmap'a, на котором производится рисование, массив графических примитивов `fFigures` и булево поле `fClrBkGnd`, определяющее необходимость очистки фона между итерациями движения и перерисовки графических примитивов. Класс включает в своем составе конструктор `Create`, производящий создание массива графических примитивов заданного типа и Bitmap'a, деструктор `Destroy`, производящий освобождение выделенной динамической памяти, методы отрисовки `Draw` и движения `DoStep` графических примитивов. Его программный код приведен ниже.

```
  { Менеджер рисования }
  TDrawManager = class
  private
    fBmp: TBitmap;
    fFigures: array of TFly;
    fClrBkGnd: Boolean;
  public
    constructor Create(Width, Height, FiguresCnt: Integer;
        FigType: TFlyClass);
    destructor Destroy(); override;

    procedure Draw();
```

```
    procedure DoStep();

    property Bitmap: TBitmap read fBmp;
    property ClearBackground: Boolean read fClrBkGnd write fClrBkGnd;
  end;

var
  Manager: TDrawManager;



{ TDrawManager }

constructor TDrawManager.Create(Width, Height, FiguresCnt: Integer;
    FigType: TFlyClass);
var
  I: Integer;
begin
  { Инициализация Bitmap'а }
  fBmp := TBitmap.Create();
  fBmp.Width := Width;
  fBmp.Height := Height;

  MAX_X := Width;
  MAX_Y := Height;

  { Инициализация массива графических примитивов }
  SetLength(fFigures, FiguresCnt);
  for I := 0 to FiguresCnt-1 do
    fFigures[I] := FigType.Create();
end;


destructor TDrawManager.Destroy();
var
  I: Integer;
begin
  inherited;

  { Удаление Bitmap'а }
  fBmp.Free;

  { Удаление фигур }
  for I := 0 to High(fFigures) do
    fFigures[I].Free();

  fFigures := nil;
end;


procedure TDrawManager.DoStep();
var
  I: Integer;
begin
  for I := 0 to High(fFigures) do
    fFigures[I].DoStep();
end;


procedure TDrawManager.Draw();
var
  I: Integer;
begin
  { Очистка }
```

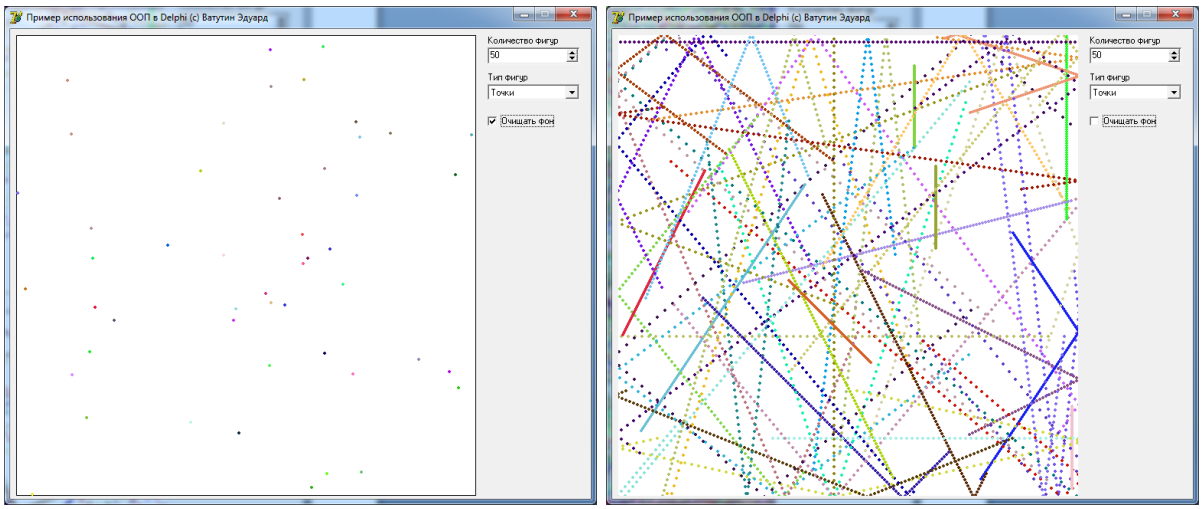
```
  if fClrBkGnd then
    with fBmp.Canvas do begin
      Pen.Color := clBlack;
      Brush.Color := clWhite;
      Rectangle(0, 0, fBmp.Width, fBmp.Height);
    end;

  { Рисование графических примитивов }
  for I := 0 to High(fFigures) do
    fFigures[I].Draw(fBmp);
end;


end.
```

Класс `TTailedFly` инкапсулирует в своем составе функционал графического примитива, представляющего движущуюся по прямой материальную точку, за которой остается «хвост» из заданного количества ее предыдущих положений. Он является наследником класса `TFly` и дополняет его полем длины «хвоста», а также переопределенными с использованием виртуального полиморфизма конструктором `Create` и методом отрисовки `Draw`. Соответствующий программный код класс приведен ниже.

```
unit TailedFly;

interface

uses
  Windows, Main, Graphics;

type
  TTailedFly = class(TFly)
  private
    procedure SetTailLength(const Value: Integer);
  protected
    fTailLng: Integer;
  public
    constructor Create(); override;
    procedure Draw(Bmp: TBitmap); override;

    property TailLength: Integer read fTailLng write SetTailLength;
  end;

implementation

{ TTailedFly }

constructor TTailedFly.Create();
begin
  fTailLng := Random(15) + 1;
  inherited;
end;

procedure TTailedFly.Draw(Bmp: TBitmap);
```

```
var
  I: Integer;
  dR, dG, dB: Double;
  Color: TColor;
  X, Y, Vx, Vy: Integer;

  procedure StepBack();
  begin
  X := X - Vx;
  if (X < 0) or (X > Manager.Bitmap.Width) then
    Vx := -Vx;

  Y := Y - Vy;
  if (Y < 0) or (Y > Manager.Bitmap.Height) then
    Vy := -Vy;
  end;

begin
  { Рисование хвоста }
  dR := (255 - GetRValue(fColor)) / fTailLng;
  dG := (255 - GetGValue(fColor)) / fTailLng;
  dB := (255 - GetBValue(fColor)) / fTailLng;

  Vx := fVx;
  Vy := fVy;
  X := fX;
  Y := fY;

  for I := 1 to fTailLng do
     with Bmp.Canvas do begin
      Color := RGB(
          Round(dR*I + GetRValue(fColor)),
          Round(dG*I + GetGValue(fColor)),
          Round(dB*I + GetBValue(fColor)),
      );
      Pen.Color := Color;
      Brush.Color := Color;

      StepBack();
      Ellipse(X-2, Y-2, X+2, Y+2);
    end;

  { Рисование головы }
  inherited;
end;

procedure TTailedFly.SetTailLength(const Value: Integer);
begin
  ASSERT(Value > 0);

  fTailLng := Value;
end;

end.
```

Примеры работы класса `TFly` приведены на рис. 1.



а) б)

Рис. 1. Примеры рисования 50 движущихся по прямым материальных точек с очисткой фона (а) и без очистки фона (б)

Примеры работы класса `TTailedFly` приведен на рис. 2.

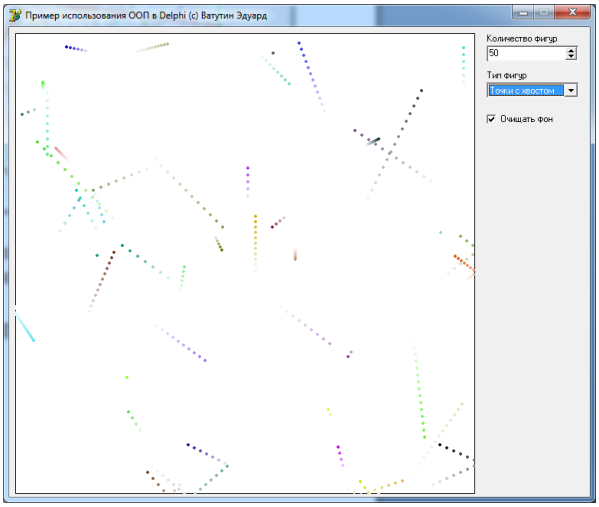


Рис. 2. Примеры рисования 50 движущихся по прямым материальных точек с «хвостом»

Реализуем функционал движения и рисования окружностей. Для создадим класс `TRound`, являющийся наследником от класса `TFly`. Он отличается от базового класса наличием поля `fRadius` (и соответствующего ему свойства `Radius`) с радиусом окружности и виртуально полиморфным методом отрисовки `Draw`. Метод движения по прямой не отличается от

движения материальной точки, поэтому он наследуется от базового класса без изменения. Соответствующий программный код приведен ниже.

```
unit Rnd;

interface

uses
  Main, Graphics;

type
  TRound = class(TFly)
  private
    procedure SetRadius(const Value: Integer);
  protected
    fRadius: Integer;
  public
    constructor Create(); override;
    procedure Draw(Bmp: TBitmap); override;

    property Radius: Integer read fRadius write SetRadius;
  end;

implementation

{ TRound }

constructor TRound.Create();
begin
  inherited;
  fRadius := Random(10) + 1;
end;

procedure TRound.Draw(Bmp: TBitmap);
begin
  with Bmp.Canvas do begin
    Pen.Color := fColor;
    Brush.Color := fColor;
    Ellipse(fX-fRadius, fY-fRadius, fX+fRadius, fY+fRadius);
  end;
end;

procedure TRound.SetRadius(const Value: Integer);
begin
  ASSERT(Value > 0);
  fRadius := Value;
end;

end.
```

Пример работы программного кода класса приведен на рис. 3.

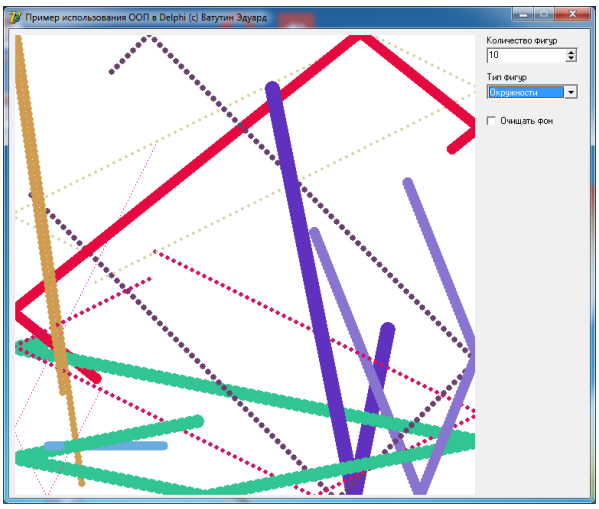


Рис. 3. Пример рисования 10 окружностей, движущихся по прямым, без очистки фона

Реализуем для окружностей функционал движения по спирали. Для этого создадим класс `TSpiral`, являющийся наследником от класса `TRound`, дополним его полями радиуса движения `fR`, текущего угла `fAngle`, угловой скорости `fW`, новым виртуальным конструктором `Create` для псевдослучайной инициализации добавленных полей, а также виртуальными методами движения `DoStep` и отрисовки `Draw`. Соответствующий программный код приведен ниже.

```
unit Spiral;

interface

uses
  Rnd, Graphics;

type
  TSpiral = class(TRound)
  protected
    fR: Integer;
    fAngle: Double;
    fW: Double;
  public
    constructor Create(); override;
    procedure DoStep(); override;
    procedure Draw(Bmp: TBitmap); override;
  end;

implementation

{ TSpiral }
```

```
constructor TSpiral.Create();
begin
  inherited;
  fR := Random(20) + 10;
  fW := Random*0.6 - 0.3;
end;

procedure TSpiral.DoStep();
begin
  inherited;

  fAngle := fAngle + fW;
  if fAngle > 2*Pi then
    fAngle := fAngle - 2*Pi;
  if fAngle < -2*Pi then
    fAngle := fAngle + 2*Pi;
end;

procedure TSpiral.Draw(Bmp: TBitmap);
var
  X, Y: Integer;
begin
  with Bmp.Canvas do begin
    Pen.Color := fColor;
    Brush.Color := fColor;

    X := Round(fX + fR*Cos(fAngle));
    Y := Round(fY + fR*Sin(fAngle));

    Ellipse(X-fRadius, Y-fRadius, X+fRadius, Y+fRadius);
  end;
end;

end.
```

Пример работы разработанного класса приведен на рис. 4.

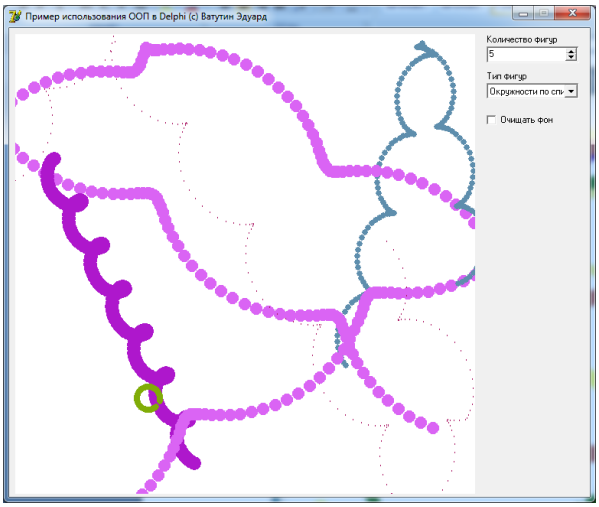


Рис. 4. Движение 5 окружностей по спиралям без очистки фона

Таким образом, в ходе выполнения практической работы была разработана объектно-ориентированная иерархия классов, включающая в своем составе 4 класса, инкапсулирующие в своем составе необходимый функционал.

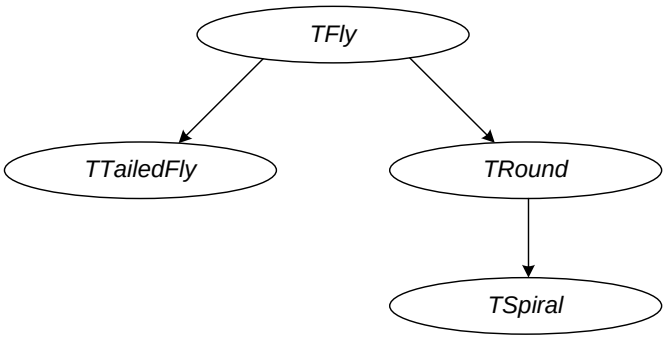


Рис. 5. Объектно-ориентированная иерархия классов

Контрольные вопросы

1. Какие 3 основные парадигмы объектно-ориентированного программирования существуют?
2. Для чего применяется наследование?
3. В чем отличие классов от объектов?
4. Какие модификаторы доступа к компонентам классов существуют?
5. В чем отличие статического и виртуального полиморфизма?
6. Что такое свойства и чем они отличаются от полей?
7. В чем отличие метаклассов от классов?
8. Что такое конструктор и деструктор?

Библиографический список

1. Емельянов С.Г., Ватутин Э.И., Панищев В.С., Титов В.С. Процедурно-модульное программирование на Delphi: учебное пособие. М.: Аргамак-Медиа, 2014. 352 с.

2. Зотов И.В., Ватутин Э.И., Борзов Д.Б. Процедурно-ориентированное программирование на C++: учебное пособие. Курск: КурскГТУ, 2008. 211 с.