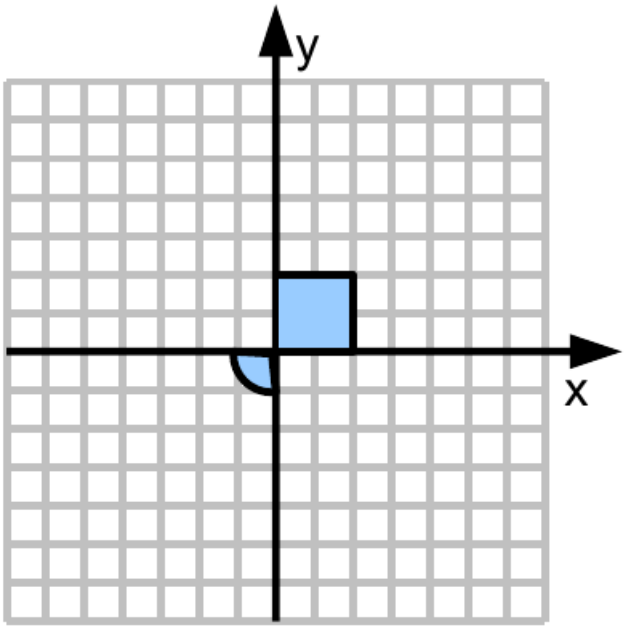
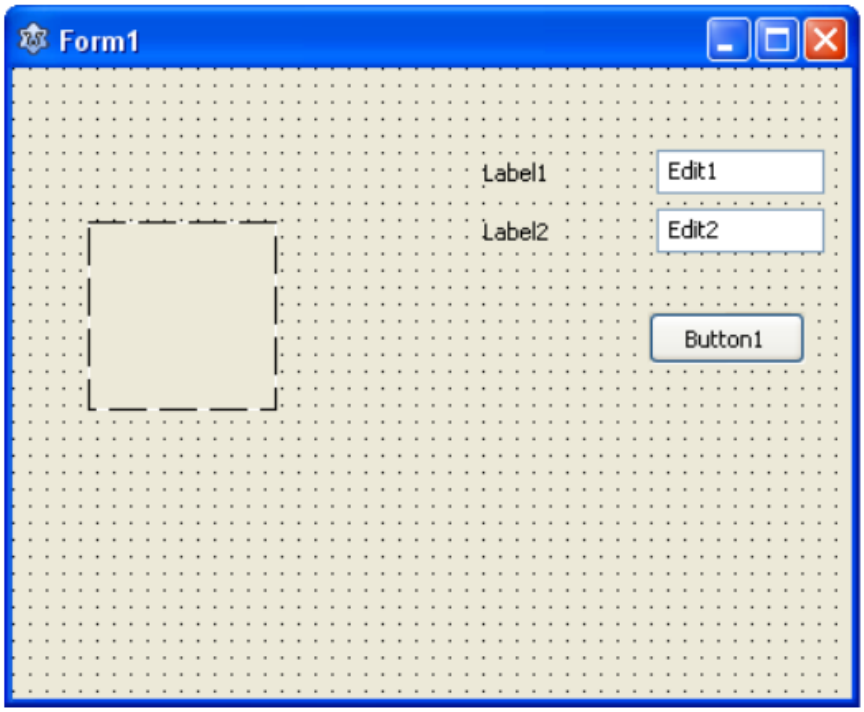
Решение задачи с помощью графического полотна Canvas

Задание: Необходимо отобразить рисунок на форме программы и дать возможность пользователю вводить координаты точек. Программа сообщает: точка с данными координатами принадлежит области или нет.

Пример к лабораторной работе данная область.

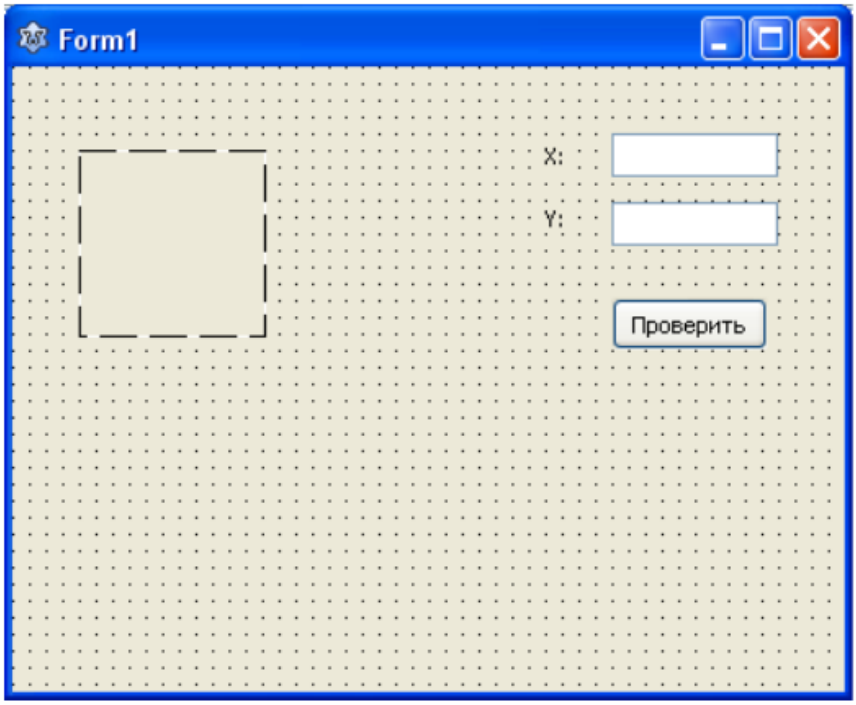


Интерфейс выглядит следующим образом:



Слева на форме компонент TImage (прямоугольник из пунктирных линий), он находится на закладке компонент Additional. Перенесем его с палитры компонентов на форму так же, как и другие компоненты. Справа: два TEdit, два TLabel и один TButton. Расставим их так, как на рисунке

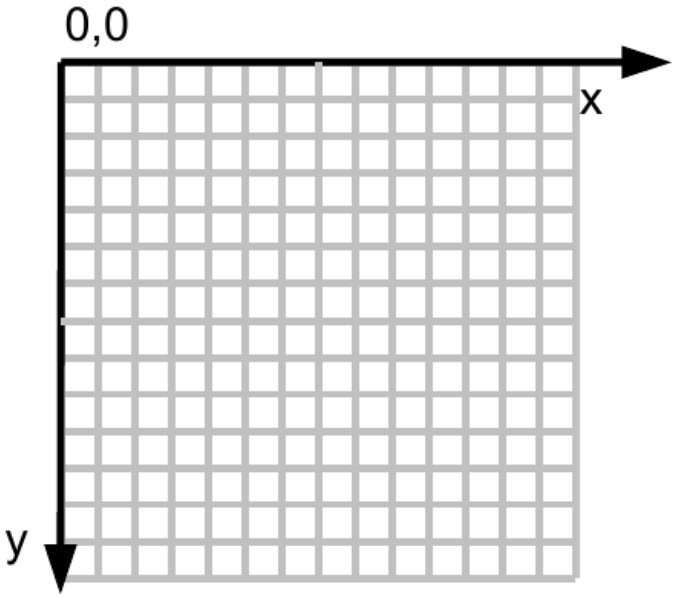
Необходимо переименовать элементы TLabel и TButton (свойство Caption) и стереть значения полей Text у компонентов TEdit. В результате получим форму



Настроим компонент «Image1» – на нем мы будем рисовать области. В инспекторе объектов найдем у него свойства «Width» и «Height» и устанавливаем их значения равными 200. Мы выбрали значение такими, что бы легче было рисовать и рассчитывать координаты.

Немного о системе координат.

У компонента «TImage», как и у всех других компонент, система координат перевернута от обычной (которой вы привыкли пользоваться). Центр координат находиться в левом верхнем углу картинки и имеет значение 0,0. Ось X — направлена горизонтально вправо, ось Y направлена вертикально вниз. Значения координат отсчитываются в пикселях (один пиксель — одна единица).



В задании центр координат находится по центру рисунка и ось Y направлена в другую сторону (вверх). Следует учесть, что значения координат в задании довольно маленькие (от -5 до 5), поэтому мы введем масштабный коэффициент и перенесем центр координат в центр картинки. Масштаб выберем равным 20:1, а центр в точке с координатами x=100; y=100. Что

идеально подходит для нашего рисунка.

Интерфейс разработан — сохраняем проект и запускаем на выполнение.

С помощью компонента «TImage» можно отобразить на форме любой рисунок. Для этого можно воспользоваться двумя способами:

1. Создать рисунок в любом графическом редакторе (Paint, Photoshop и др.) и загрузить получившийся рисунок (файл рисунка) в компонент Image1 (формат BMP подойдет, хотя можно использовать и другие).
2. Воспользоваться примитивными функциями рисования и последовательно создать рисунок из таких элементов как: линии, точки, дуги, окружности, эллипсы и т. д.

Первый способ лучше подходит для таких рисунков как фотографии, а второй для схем.

Мы воспользуемся вторым способом.

Теперь решим когда необходимо отображать рисунок нашей области? Конечно же в самом начале выполнения программы, т. к. пользователь должен сразу видеть область, а не выполнять какие то действия по ее отображению.

Наиболее подходящее место по отображения области это процедура создания формы. Когда форма создается (ее границы, метки, поля ввода, кнопки и др. компоненты) мы рисуем область.

Что бы выбрать обработчик создания формы сделаем двойной клик в свободной области формы (где нет других компонентов). В результате получим следующий обработчик.

```
.   { TForm1 }
35
.  procedure TForm1.FormCreate(Sender: TObject);
.  begin
38
.  end;
```

Между элементам Begin и End добавим код, который будет рисовать область.

Для рисования необходимо обратиться к тому компоненту, на котором мы будем рисовать, в данном случае компонент Image1. У компонента имеется много свойств, с некоторыми уже познакомились. Свойство отвечающее за рисование примитивов называется Canvas.

Рассмотрим как происходит прорисовка.

Существует ряд процедур у свойства Canvas которые рисуют примитивные объекты. Например для отображения линии необходимо выбрать

процедуру Line и передать ей 4 координаты (координаты начала и конца линии).

На экране после вызова процедуры отобразится линия. Для построения эллипса необходимо вызвать процедуру Ellipse с 4-мя координатами. Это координаты левого верхнего и правого нижнего угла прямоугольника, в который будет вписан эллипс (направления осей эллипса совпадают с направлением осей координат).

Canvas имеет два свойства отвечающие за цвета: Кисть (Brush) и Карандаш (Pen). Все линии объектов рисуются свойствами карандаша: цвет линии, толщина, стиль и т. д., а все объемные части объектов закрашиваются свойствами кисти: цвет заливки, штриховка и др. Например внутренняя часть эллипса будет закрашена свойствами кисти, а внешняя граница нарисована свойствами карандаша.

В среде Lazarus нет одной процедуры, которая сразу нарисовала бы такой сложный рисунок как в задании. Есть несколько примитивных функций с помощью которых можно последовательно (по правилу художника) нарисовать практически любой рисунок.

Разделим рисунок на 3 области, которые будем рисовать:

1 — четверть окружности;

2 — прямоугольник;

3 — оси координат.

Прямоугольник и линии можно нарисовать средствами Canvas вызвав по одной процедуре, а четверть окружности — нет. Для отображения четверти окружности мы нарисуем всю окружность, а лишнее сотрем.

Как рассчитывать координаты? Воспользуемся следующими формулами для перевода координат из физических в экранные.

X = X*20 + 100

Y =100 – Y*20

Отобразим в центре рисунка окружность. Добавим следующий код.

```
35
 .  procedure TForm1.FormCreate(Sender: TObject);
 .  begin
 .    Image1.Canvas.Ellipse(80,80,120,120);
 .  end;
40
```

Сохраним проект и запустим на выполнение.

По умолчанию карандаш имеет цвет черный, а кисть белый. Поэтому окружность белого цвета — она закрашена белой кистью.

Для закраски фона воспользуемся белым прямоугольником размером равным размеру компонента Image1. Очистку фона надо вызвать до прорисовки окружности, что бы не стереть окружность. Добавим следующую строчку выше

```
35
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Image1.Canvas.Rectangle(0,0,200,200);
    Image1.Canvas.Ellipse(80,80,120,120);
40 end;
```

Сохраним проект и запустим на выполнение.

Немного о цветах: Все чистые цвета и многие оттенки цветов могут быть представлены в виде трех составляющих: красного, зеленого и синего. В ОС Windows эта комбинация трех цветов называется RGB по первым буквам соответствующих цветов на английском (Red, Green, Blue). Каждая составляющая берется в диапазоне от 0 до 255 и чем больше число, тем сильнее выражен соответствующий цвет. Пример: для красного цвета надо красную составляющую повысить на максимум, а остальные убрать на минимум (255,0,0). В среде Lazarus присутствует константы цветов — те которыми мы привыкли пользоваться в повседневной жизни но на английском языке. Например для красного цвета это константа clRed. Константа состоит из префикса «cl» (сокращение от английского Color) и самого названия цвета Red (красный на английском языке).

Закрасим нашу окружность зеленым цветом. Для этого необходимо до рисования окружности назначить зеленый цвет кисти.

```
35
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Image1.Canvas.Rectangle(0,0,200,200);
    Image1.Canvas.Brush.Color:=clGreen;
40  Image1.Canvas.Ellipse(80,80,120,120);
  end;
```

Сохраним проект и запустим на выполнение.

По заданию необходима не вся окружность, а только ее часть (четверть). Что бы убрать лишние части окружности — изменим кисть не белую и карандаш то же выберем белый. Прорисуем поверх зеленого круга белые прямоугольники. Допишем следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Rectangle(0,0,200,200);
  Image1.Canvas.Brush.Color:=clGreen;
  Image1.Canvas.Ellipse(80,80,120,120);
  Image1.Canvas.Brush.Color:= clWhite;
  Image1.Canvas.Pen.Color:=clWhite;
  Image1.Canvas.Rectangle(80,80,120,100);
  Image1.Canvas.Rectangle(100,100,120,120);
end;
```

Сохраним проект и запустим на выполнение.

Добавим следующий код.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.Rectangle(0,0,200,200);
  Image1.Canvas.Brush.Color:=clGreen;
  Image1.Canvas.Ellipse(80,80,120,120);
  Image1.Canvas.Brush.Color:= clWhite;
  Image1.Canvas.Pen.Color:=clWhite;
  Image1.Canvas.Rectangle(80,80,120,100);
  Image1.Canvas.Rectangle(100,100,120,120);
  Image1.Canvas.Pen.Color:=clBlack;
  Image1.Canvas.Brush.Color:=clGreen;
  Image1.Canvas.Rectangle(100,60,140,100);
  Image1.Canvas.Line(0,100,200,100);
  Image1.Canvas.Line(100,0,100,200);
end;
```

Разберитесь с новыми командами и что они делают?

Сохраним проект и запустим на выполнение.

Программа рисует область, а проверять координаты пока не умеет. Добавим эту возможность в программу. Необходимо написать обработчик на кнопку «Проверить». Закроем запущенную программу и в редакторе формы сделаем двойной «клик» на кнопке «Проверить».

Для проверки понадобятся две вспомогательные переменные в которых будем хранить координаты, которые ввел пользователь в поля «X:» и «Y:». Координаты вещественные (могут иметь дробную часть), объявим их как Real. Запишем в эти переменные значения из соответствующих полей.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x,y : Real;
begin
  x := StrToFloat(Edit1.Text);
  y := StrToFloat(Edit2.Text);
end;
```

Запустив проект мы не заметим никаких изменений в работе программы.

Необходимо проанализировать значения координат с помощью оператора выбора «IF». Воспользуемся логическими операциями AND, OR, NOT и операциями сравнения «>» «<» «>=» «<=» «<>» «=». Анализ координат разделим на анализ в каждой отдельной фигуре (четверти окружности и прямоугольнике). Если точка принадлежит хотя бы одной из фигур, то она принадлежит нашей области.

Точка принадлежит прямоугольнику, когда координата «X» в пределах от левой до правой грани, а координата «Y» от верхней до нижней. Здесь уже используются координаты физические, а не экранные (те что заданы на рисунке в задании). Для проверке четверти окружности необходимо знать уравнение всей окружности:

$$(X-X_0)^2+(Y-Y_0)^2=R^2$$

где: X,Y - координаты проверяемой точки;

X0 , Y0 - координаты центра окружности;

R – радиус окружности.

Если в уравнении стоит знак «=» то точка с координатами (X, Y) находятся на окружности, если поставить знак «<» то внутри окружности, а если «>» то вне окружности. Воспользуемся знаком «<» (строго меньше, т. к. граница не входит в область — вариант первый — нечетный) и определим значения X0 и Y0 .

Центр окружности находится в точке с координатами 0,0 тогда X0 =0 и Y 0 =0 .

По заданию не вся окружность, а только четверть, значит дополнительно ограничим нашу окружность одной четвертью. Фигуры (четверть окружности и прямоугольник) между собой объединяются с помощью оператора OR (или), а условия принадлежности фигуре с помощью AND (и).

Оператор выбора (IF) может содержать один или два (как в данном случае) вложенных в него оператора. Первый оператор находящийся после слова «Then» выполнится если условие выполнится, а второй ели условие не выполнится. Обо одновременно не могут выполнится, и ни одни тоже не может выполнится. Хотя бы один из операторов (после слова «Then» или «Else») всегда выполняется, но какой зависит от условия. Если условие выполняется будем выдавать (при помощи функции ShowMessage) текст «Точка принадлежит области», а если условие не выполнится, то «Точка НЕ принадлежит области».

Добавим следующий код.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x,y : Real;
begin
  x := StrToFloat(Edit1.Text);
  y := StrToFloat(Edit2.Text);
  if ((x>0)and(x<2)and(y>0)and (y<2))or
     ((x*x+y*y<4)and(x<0)and(y<0)) then
     ShowMessage('Точка принадлежит  области') else
     ShowMessage('Точка НЕ принадлежит области');
end;
```

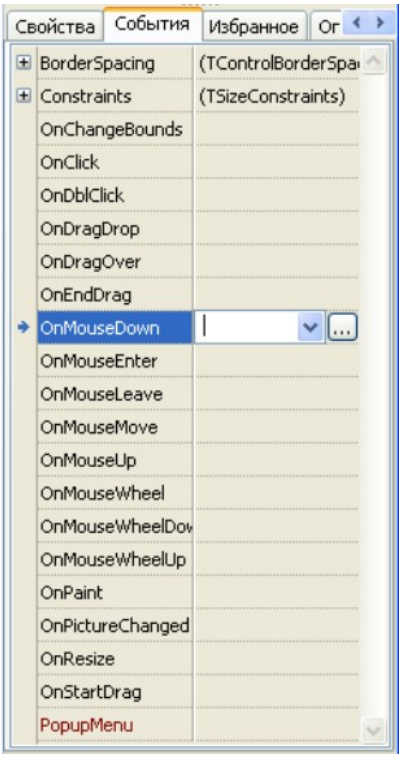
Сохраним проект и запустим на выполнение. В соответствующие поля введем координаты (например 1, 1) и нажмем кнопку проверить.

Программа почти закончена, но необходимо добавить в нее еще одну небольшую функциональность, которая значительно облегчит проверку правильности составления условия (оператора IF) на принадлежность точек области. Пользователю удобней не вводить координаты в соответствующие поля ввода данных, а «кликать» «мышью» на рисунке. Место «клика» программа проанализирует и занесет в соответствующие поля значения координат.

Получения координат происходит в момент «клика» по компоненту Image1. Событие «OnClick» (связанное с Image1) не подойдет. Хотя оно происходит в момент «клика», но оно не содержит значений координат «мыши» в момент клика, которые необходимы для анализа. Более подходящие события: OnMouseDown и OnMouseUp.

Они содержат координаты курсора мыши в момент «клика». Первое происходит в момент нажатия на кнопку «мыши» - Down, а второе в мент отпускания кнопки мыши - Up. Эти события происходят в разные моменты времени и могут содержать разные координаты. Можно отпустить кнопку «мыши» не в том месте где нажали ее.

Более подходящее событие для анализа координат «мыши» это OnMouseDown. Найдем его в «Инспекторе объектов», **предварительно выбрав компонент Image1**, на вкладке «события».



Список возможных привязываемых событий пуст, но справа есть кнопка «...» нажмем на нее и получим заготовку события OnMouseDown.

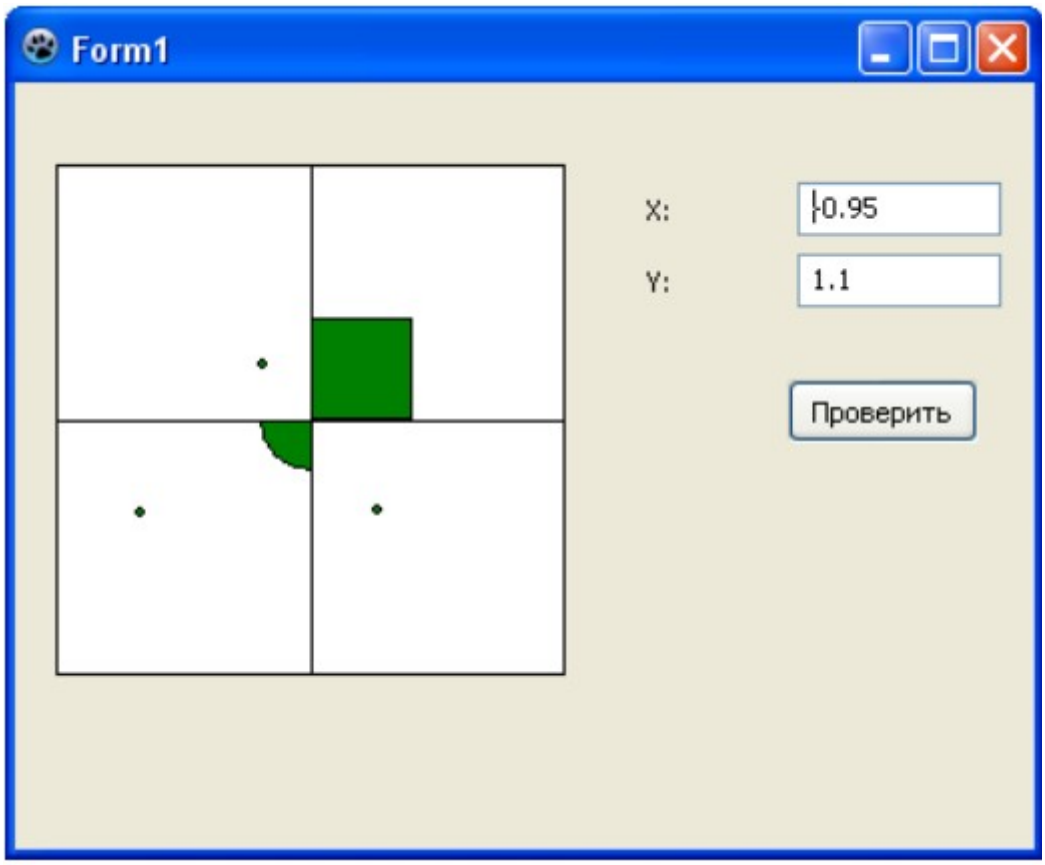
```
55  procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
    .    Shift: TShiftState; X, Y: Integer);
    .  begin
    .
    .  end;
```

Добавим необходимый код

```
55  procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
    .    Shift: TShiftState; X, Y: Integer);
    .  begin
    .    Edit1.Text:=FloatToStr((X-100)/20);
    .    Edit2.Text:=FloatToStr((Y-100)/(-20));
60   Image1.Canvas.Ellipse(x-2,y-2,x+2,y+2);
    .  end;
```

Сохраним проект и запустим на выполнение.

Сделав несколько «кликов» на компоненте Image1 получим примерно следующий вид программы.



Работа над демонстрационном примером закончена. Воспользуйтесь этой программой и нарисуйте свою область не забыв изменить процедуру проверки координат. Интерфейс программы менять нет необходимости, а все изменения производятся в процедурах «Button1Click» и «FormCreate».

Список графических примитивов Delphi (Lazarus).

Класс TCanvas — сердцевина графической подсистемы Delphi (Lazarus). Он объединяет в себе и "холст" (контекст конкретного устройства GDI), и "рабочие инструменты" (перо, кисть, шрифт) и даже "подмастерьев" (набор функций по рисованию типовых геометрических фигур).

Канва не является компонентом, но она входит в качестве свойства во многие другие компоненты, которые должны уметь нарисовать себя и отобразить какую-либо информацию.

Для рисования канва включает в себя шрифт, перо и кисть:

```
(pb) property Font: TFont ;  {TFont: Charset, Color, Style, Size}
(Pt) property Pen: TPen; {TPen: Color, Mode, Style, Width}
(Pb) property Brush: TBrush;  {TBrush: Bitmap, Color, Style}
```

Кроме того, можно рисовать и поточечно, получив доступ к каждому пикселу. Значение свойства

```
property Pixels[X, Y: Integer]: TColor;
```

соответствует цвету точки с координатами (X,Y).

Класс TCanvas

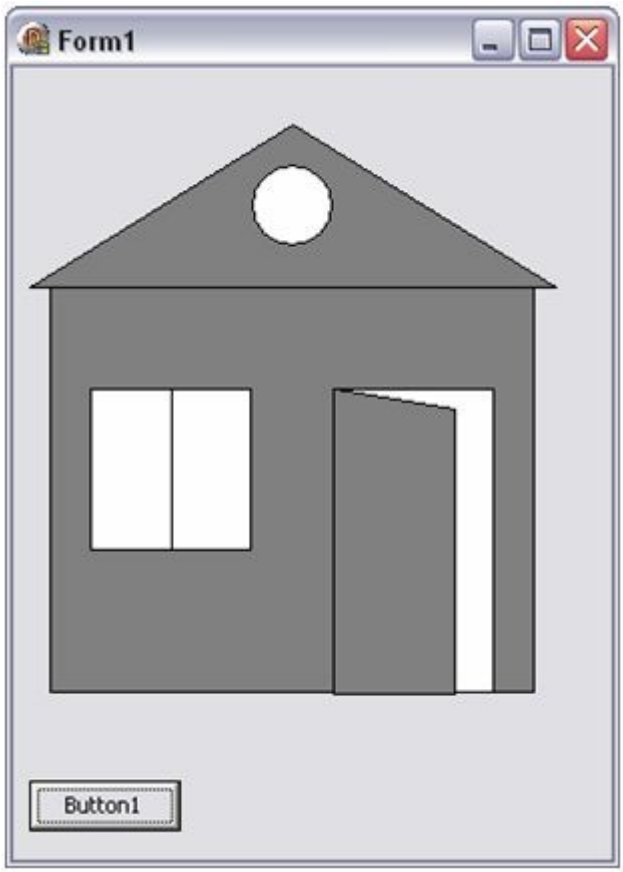
`procedure Arc (X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);`	Метод рисует сегмент эллипса. Эллипс определяется описывающим прямоугольником (X1,Y1) — (X2,Y2). Начальная точка сегмента лежит на пересечении эллипса и луча, проведенного из его центра через точку (X3,Y3). Конечная точка сегмента лежит на пересечении эллипса и луча, проведенного из его центра через точку (X4,Y4). Сегмент рисуется против часовой стрелки.
`procedure Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);`	Рисует хорду и заливает отсекаемую ею часть эллипса. Эллипс, начальная и конечная точки определяются, как в методе Arc.
`procedure Ellipse(X1, Y1, X2, Y2: Integer);`	Рисует и закрашивает эллипс, вписанный в прямоугольник (X1,Y1) — (X2,Y2).
`procedure MoveTo(X, Y: Integer);`	Перемещает текущее положение пера (свойство PenPos) в точку (X,Y).
`procedure LineTo(X, Y: Integer);`	Проводит линию текущим пером из текущей точки в (X,Y).
`procedure BrushCopy(const Dest: TRect; Bitmap: TBitmap; const Source: TRect; Color: TColor);`	Производит специальное копирование. Прямоугольник Source из битовой карты Bitmap копируется в прямоугольник Dest на канве; при этом цвет Color заменяется на цвет текущей кисти (Brush.Color).
`procedure CopyRect(const Dest: TRect; Canvas: TCanvas; const Source: TRect);`	Производит копирование прямоугольника Source из канвы Canvas в прямоугольник Dest в области самого объекта.
`procedure FillRect(const Rect: TRect);`	Производит заливку прямоугольника (текущей кистью).
`procedure Draw(X, Y: Integer; Graphic: TGraphic);`	Осуществляет рисование графического объекта Graphic
`procedure StretchDraw(const Rect: TRect; Graphic: TGraphic);`	Осуществляет рисование объекта Graphic в заданном прямоугольнике Rect. Если размеры их не совпадают, Graphic масштабируется.
`procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle: TFillStyle); TFillStyle = (fsSurface, fsBorder);`	Производит заливку области текущей кистью. Процесс начинается с точки (X,Y). Если режим FillStyle равен fsSurface, то он продолжается до тех пор, пока есть соседние точки с цветом Color. В режиме fsBorder закрашивание, наоборот, прекращается при выходе на границу с цветом Color.
`procedure Pie (X1, Y1, X2,`	Рисует сектор эллипса, описываемого

`Y2, X3, Y3, X4, Y4: Integer);`	прямоугольником (X1,Y1) — (X2,Y2). Стороны сектора лежат на лучах, проходящих из центра эллипса через точки (X3.Y3) и (X4,Y4).
`procedure Polygon(const Points: array of TPoint);`	Строит многоугольник, используя массив координат точек Points. При этом последняя точка соединяется с первой и внутренняя область закрашивается. Polygon ([Point(10,10), Point(30,30),Point(20,40)])
`procedure Polyline(const Points: array of TPoint);`	Строит ломаную линию, используя массив координат точек Points.
`procedure Rectangle(X1, Y1, X2, Y2 : Integer);`	Рисует прямоугольник с диагональю заданной координатами (X1,Y1) и (X2,Y2).
`procedure RoundRect (X1, Yl, X2, Y2, XW, YH: Integer);`	Рисует прямоугольник с закругленными углами. Координаты вершин — те же, что и в методе Rectangle. Закругления рисуются как сегменты эллипса с размерами осей по горизонтали и вертикали XW и YH.
`function TextHeight(const Text: string): Integer;`	Возвращает высоту строки Text в пикселах.
`function TextWidth(const Text: string): Integer;`	Возвращает ширину строки Text в пикселах.
`procedure TextOut(X, Y: Integer; const Text: string);`	Производит вывод строки Text. Левый верхний угол помещается в точку канвы (X,Y).
`procedure TextRect(Rect: TRect; X, Y: Integer; const Text: string);`	Производит вывод текста с отсечением. Как и в TextOut, строка Text выводится с позиции (X,Y); при этом часть текста, лежащая вне пределов прямоугольника Rect, отсекается и не будет видна.
`property PenPos: TPoint;`	Содержит текущую позицию пера канвы (изменяется посредством метода MoveTo).

Пример

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Paintbox1.Canvas do begin
    Brush.Color := clGray;
    Rectangle(10,100,250,300);  // корпус
    Polygon([Point(0,100),Point(130,20),Point(260,100)]); // крыша
    Brush.Color := clWhite;
    Ellipse(110,40,150,80); // чердак
    Rectangle(30,150,110,230);   // окно
    MoveTo(70,150);
    LineTo(70,230);
    Rectangle(150,300,230,150);  // дверь
    Brush.Color := clGray;
    Polygon([Point(150,300),Point(150,150),Point(210,160),Point(210,300)]);

  end;
end;
```


Form1
Button1