МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

«Утверждаю»
Зав. кафедрой «МО и ПЭВМ»
Макарычев П.П.
«____»_____2017 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине «Программирование»

на тему:

«Создание приложения, позволяющего создавать некоторые типы фигур»

Автор работы: Фрунзе П.М.

Направление: 09.03.04

Группа: 16ВП1

Руководитель работы: Гурьянов Л.В.

Работа защищена «__» ____ 2017 г. Оценка _____

Пенза 2017 г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Целью работы является создание приложения с графическим пользовательским интерфейсом, которое позволяет рисовать некоторые типы фигур.

Приложение должно:

1. Предоставлять функцию «умного» создания для каждого типа фигуры (фигура создается в указанном центре с необходимыми параметрами, численные значения которых задаются пользователем в явном виде).
2. Предоставлять функцию «быстрого» создания для каждого типа фигуры (фигура вписывается в область, обозначенную пользователем).
3. Уметь обновлять изображение.
4. Хранить все созданные экземпляры фигур в структуре типа «список».
5. Предоставлять функцию очистки изображения, которая также в обязательном порядке очищает список созданных фигур.
6. Поддерживать следующие типы фигур: точка, линия, прямоугольник, круг, круг с крестом.

Оглавление

ГЛАВА 1. АНАЛИЗ ТРЕБОВАНИЙ

1.1. Анализ предметной области

Модель предметной области расположена на рис. 1.

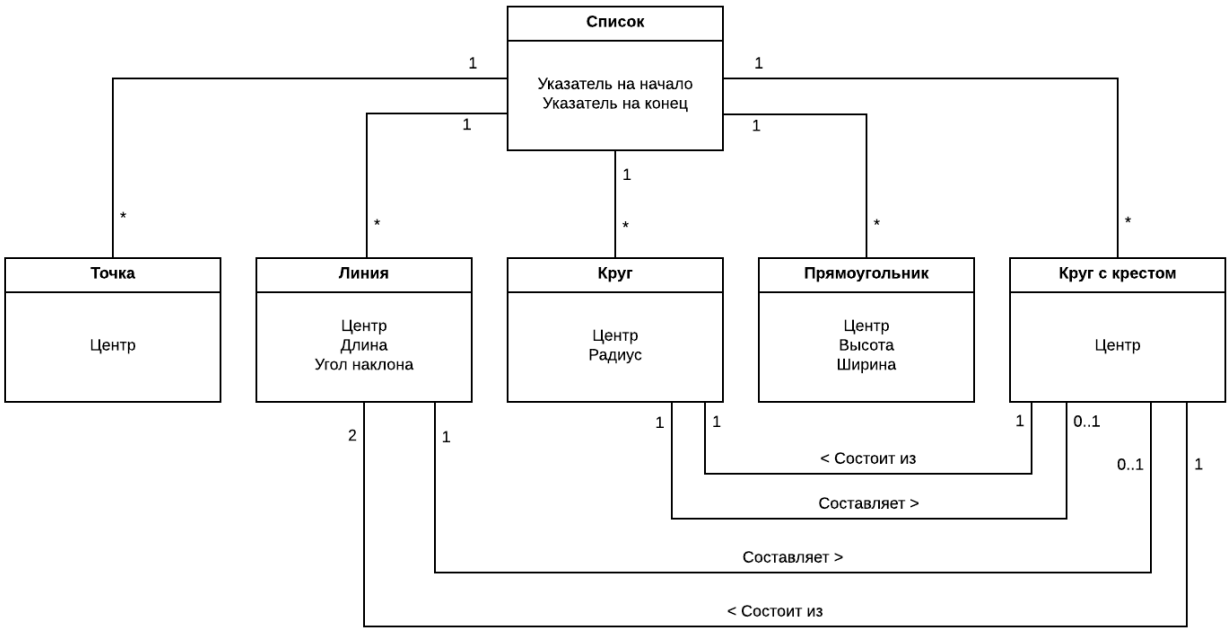

Рисунок 1 – Модель предметной области

1.2. Анализ функциональных требований

На рис. 2 представлена диаграмма вариантов использования.

Диаграмма, расположенная на рис. 2 уточняет вариант использования «Выбрать фигуру»

На рис. 4 демонстрируется сценарий выполнения варианта «Умное создание».

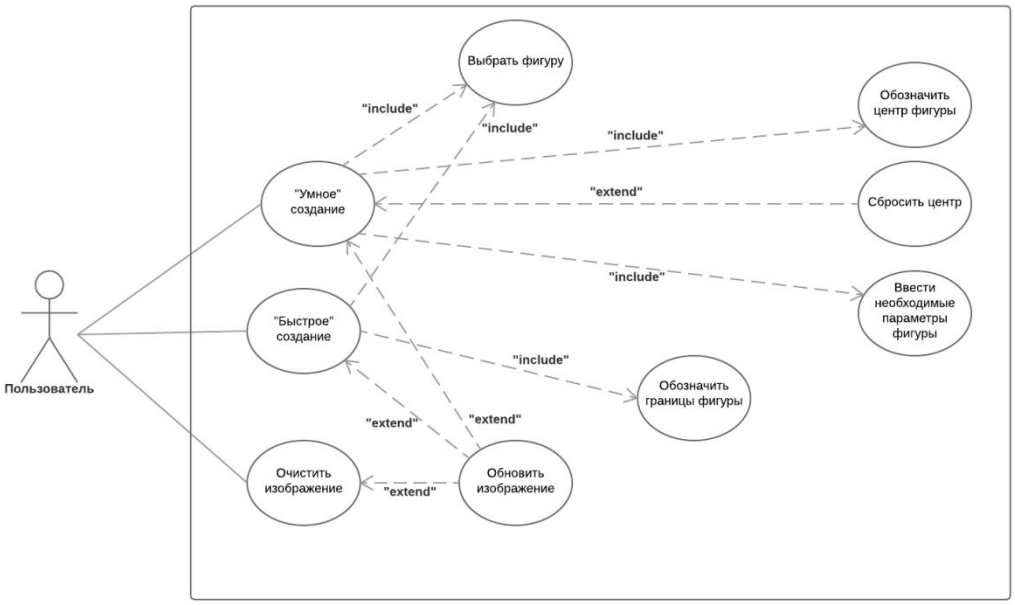

Рисунок 2 – Диаграмма вариантов использования

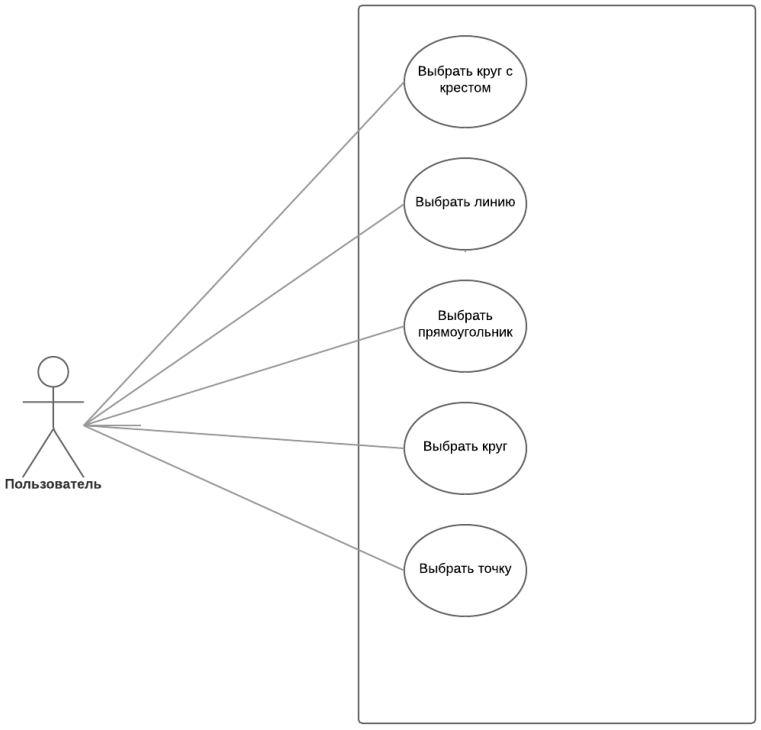

Рисунок 3 – Вариант "Выбрать фигуру"

Наименование: «Умное» создание фигуры

ID: 1

Краткое описание: система рисует выбранную фигуру с указанным центром и введенными пользователем параметрами

Действующие лица: пользователь, система

Предусловия: запуск программы пользователем

Основной поток:

1. Пользователь выбирает фигуру, которую он хочет нарисовать, в списке с доступными типами фигур.
2. Пользователь указывает, где должен располагаться центр новой фигуры.
3. Пользователь выбирает вариант использования «Умное создание».
4. ПОКА не все параметры фигуры инициализированы:
    4.1. Система запрашивает ввод значения параметра, необходимого для создания фигуры.
    4.2. Система осуществляет проверку ввода на предмет корректности.
    4.3. ЕСЛИ ввод корректен:
        4.3.1. Система инициализирует параметр фигуры заданным пользователем значением.
    4.4. ИНАЧЕ:
        4.4.1. Система выводит сообщение об ошибке.
        4.4.2. Система инициализирует параметр фигуры значением по умолчанию.
5. Система создает экземпляр фигуры.
6. Система помещает фигуру в список.
7. Система перерисовывает изображение.

Постусловия: выбранная пользователем фигура создана, добавлена в список и отображена на области рисования.

Рисунок 4 – Сценарий выполнения варианта использования

ГЛАВА 2. ПРОЕКТИРОВАНИЕ

2.1. Уточнение вариантов использования

На этапе проектирования было принято решение выделить «систему приложения» как отдельную сущность, которая реализует необходимый функционал. Сущность «графический интерфейс» необходима для обеспечения передачи запросов от пользователя к системе и наоборот. Более детально эту концепцию можно рассмотреть на диаграмме последовательности (рис. 5), которая уточняет ранее предоставленный сценарий.

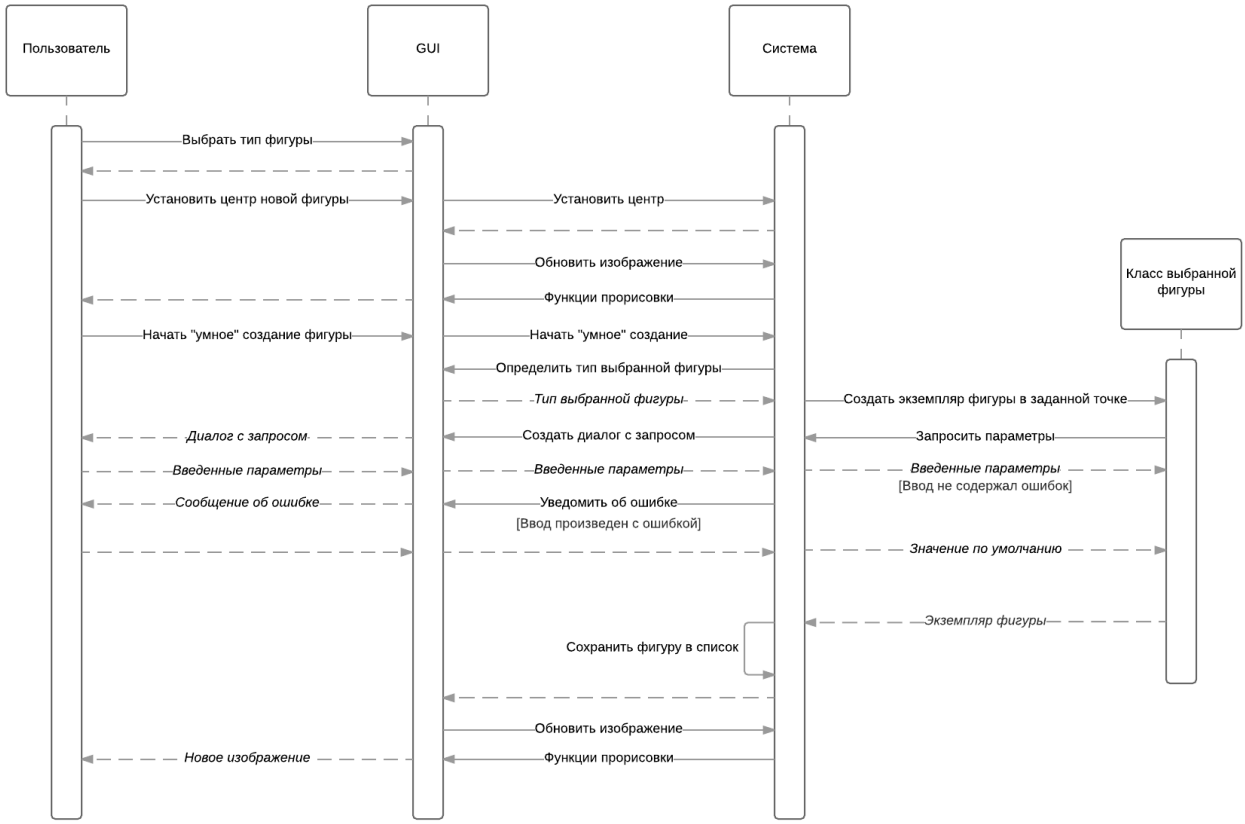

Рисунок 5 – Диаграмма последовательности

На этой же диаграмме можно проследить и другую идею: система не должна «знать», какие именно параметры необходимы для инициализации фигуры, пока она не обратится к классу данной фигуры с запросом «Создать

экземпляр». Придерживаясь этой концепции, можно будет добавлять новые типы фигур и отлаживать работу существующих, не меняя структуры самой системы, что способствует «безопасному» расширению функционала приложения.

2.2. Проектирование пользовательского интерфейса

Опираясь на ранее составленные диаграммы вариантов использования (см. рис. 2-3), был спроектирован следующий дизайн интерфейса (рис. 6):

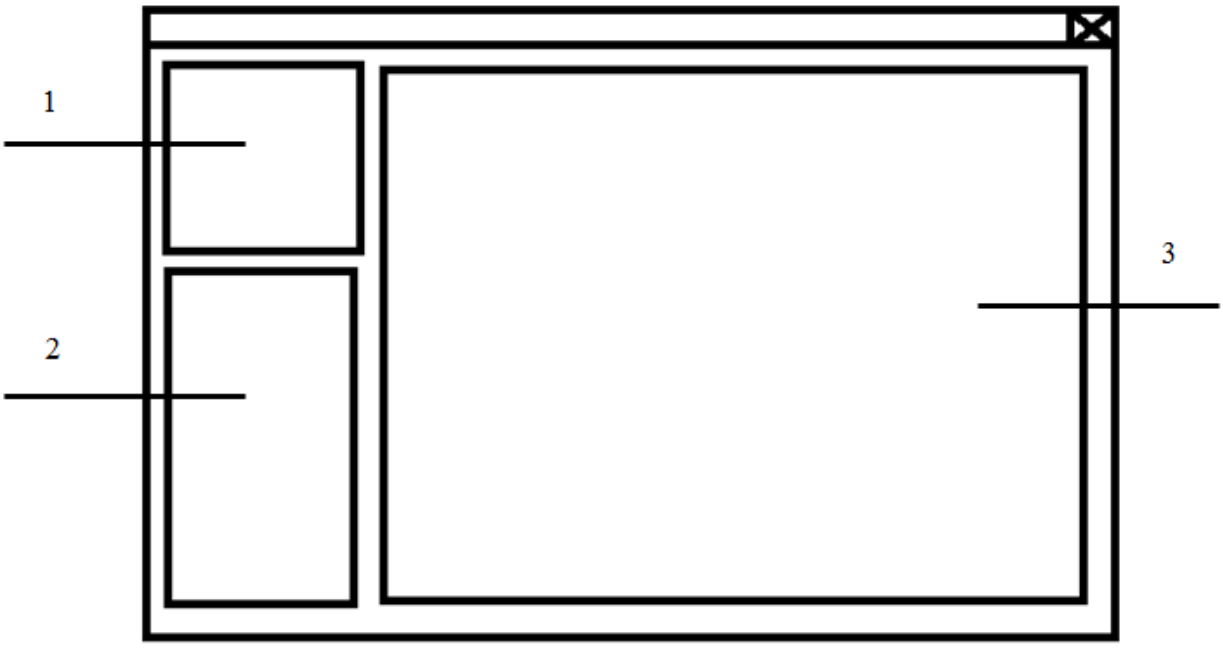

Рисунок 6 – Общий дизайн

1. Область команд: здесь будут находиться кнопки, реализующие варианты использования «Умное создание», «Сбросить центр», «Очистить изображение».
2. Список с доступными типами фигур, которые пользователь может нарисовать. Для выбора фигуры пользователю будет необходимо сделать щелчок по строке с названием фигуры, которую он хочет создать.

3. Холст: здесь будут отображаться все созданные пользователем фигуры. Также этот элемент будет использоваться для создания фигур. Это будет происходить следующим образом:

Для *«быстрого создания»* пользователь будет использовать холст, чтобы обозначить границы, в которые будет вписана новая фигура (рис. 7).

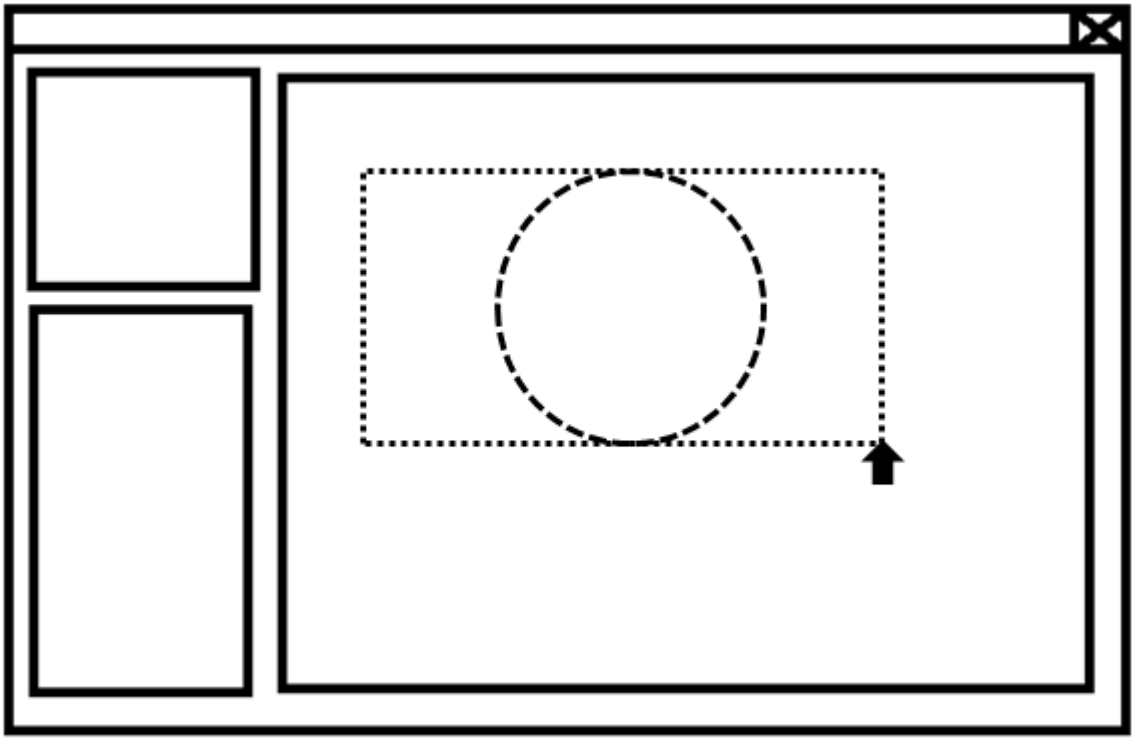

Рисунок 7 – Дизайн быстрого создания

Все фигуры обладают центром, поэтому этап инициализации центра, который является необходимым при *«умном создании»* можно обобщить для всех фигур. Вместо ввода числовых значений координат центра предоставим пользователю возможность выбрать центр новой фигуры, щелкнув по холсту в желаемом местоположении центра. Дизайн пометки центра можно увидеть на рис. 8.

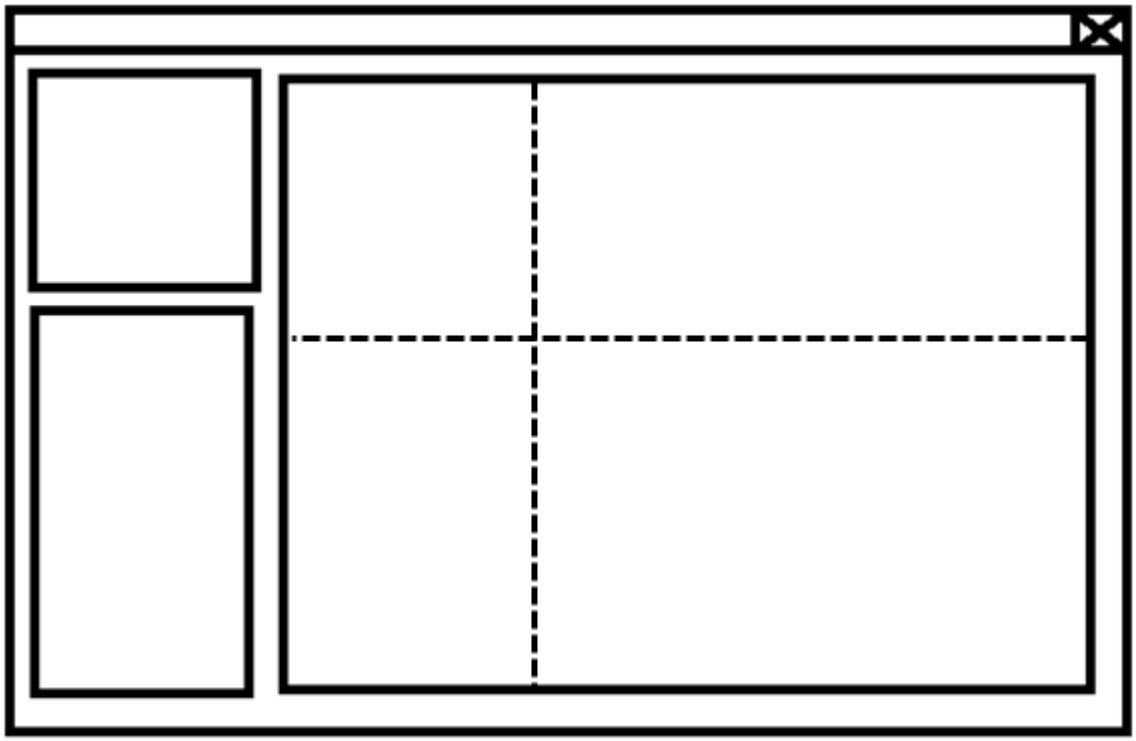

Рисунок 8 – Дизайн обозначения центра

Ввод остальных параметров будет осуществляться с помощью диалогов, которые будут вызваны при нажатии на кнопку «Умное создание». Предоставим пользователю возможность отменить создание фигуры, отказавшись от ввода. Дизайн диалога можно увидеть на рис. 9.

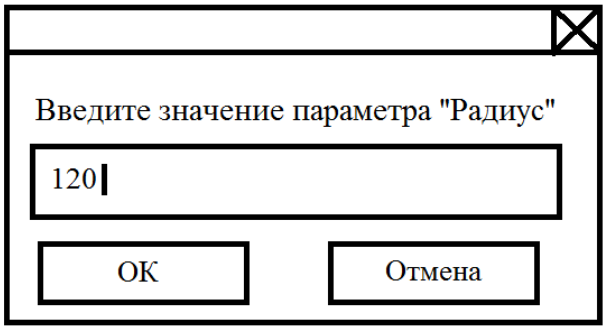

Рисунок 9 – Дизайн диалогового окна

Следует заметить, что во время проектирования интерфейса было принято решение отказаться от возможности явного вызова выполнения варианта

использования «Обновить изображение» пользователем. Это связано с тем, что исполнение этой команды бессмысленно, если на изображении ничего не поменялось. Вместо этого приложение всякий раз автоматически будет обновлять изображение, когда это необходимо.

2.3. Проектирование программных средств

Для реализации поставленной задачи будем пользоваться технологией ООП. Фигуры, поведение которых необходимо реализовать, можно объединить в иерархическую систему классов – каждый из пяти типов фигур, представленных в модели предметной области, может быть наследником базового абстрактного класса «Фигура». В рамках данной работы это решение имеет определенные достоинства:

1. Использование механизма *наследования* позволит вынести атрибут «Центр», которым обладает каждый конкретный тип фигуры, в базовый класс, избавляя от необходимости определять его в каждом классе отдельно. Также наследование обеспечивает возможность использования другого механизма – *полиморфизма*.
2. *Полиморфные методы* дадут возможность реализовать концепцию, при которой структура системы приложения не зависит от того, какие именно фигуры были реализованы в данный момент. Вместо сбора сведений об индивидуальных способах взаимодействия с каждым классом отдельно, система будет подключать все доступные классы фигур из специально созданного модуля и обращаться к ним через единый интерфейс, объявленный в базовом абстрактном классе.
3. *Инкапсуляция данных* обеспечит безопасность работы с классами фигур. Следуя концепции инкапсуляции данных, будет невозможно:
    3.1. нарушить механизм работы существующих классов при добавлении нового кода;

3.2. влиять на поведение фигур в степени, большей, чем это позволяет набор методов, которые и реализуют все необходимые варианты использования фигуры.

Для реализации сущности «Список», которая должна хранить в себе экземпляры всех созданных пользователем фигур, будет также удобно пользоваться технологией ООП. Процедуры работы с этой структурой являются в определенной степени сложными и небезопасными[1], поэтому будет полезно инкапсулировать их реализацию в специально созданном классе.

Для хранения доступных классов[2] фигур удобно пользоваться структурой типа «массив», потому что в рамках данной задачи это решение имеет определенное преимущество в плане оптимизации[3].

На рис. 10 изображена диаграмма классов.

---

[1] Для своей реализации списки требуют использования указателей, неосторожная работа с которыми может повлечь за собой определенный ряд ошибок.

[2] Здесь имеется ввиду хранение именно самих классов, а не их экземпляров. Реализация этой идеи в разных языках может выглядеть по-разному. Например, в языке Delphi, который будет использоваться в данной работе, существует довольно удобный механизм хранения указателей на классы.

[3] Массивы предоставляют возможность произвольного доступа к своим элементам. В процессе работы приложения порядок обращения к элементам структуры, которая хранит классы, будет гарантированно неизвестным заранее, так как это зависит от действий пользователя. Недостаток же массивов – затратное удаление и добавление элементов – не будет проявляться, так как ни одна из этих операций не планируется быть исполненной в процессе выполнения программы.

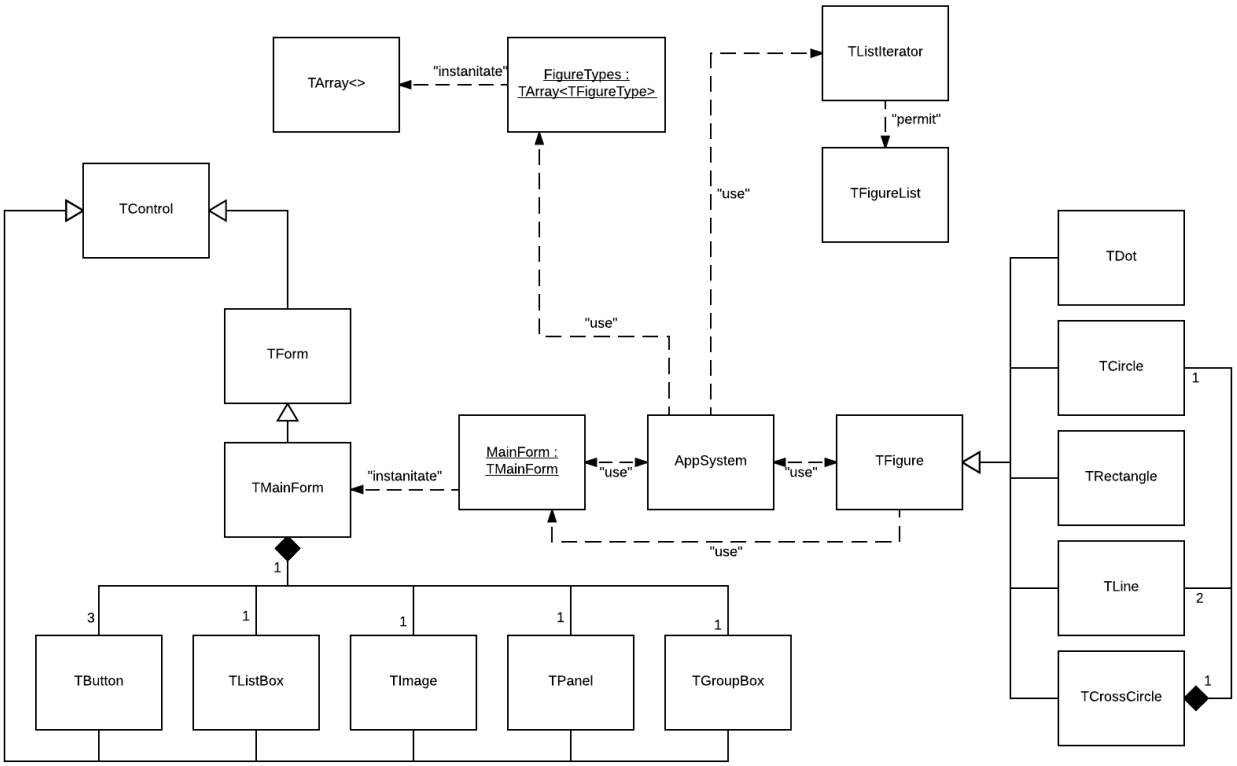

Рисунок 10 – Диаграмма классов

Продемонстрируем спецификацию класса на примере AppSystem:

Сущность: система приложения.

Назначение: реализует функциональные требования к приложению.

Поля:

- Созданные фигуры : список фигур

- Центр фигуры для умного создания : точка *(матем.)*

- Первая точка, определяющая границы при быстром создании : точка *(матем.)*

- Состояние установки центра для умного состояние *(установлен/нет)* : булев.

+ Состояние быстрого создания *(производится в данный момент/нет)* : булев.

+ Тип выбранной фигуры : тип фигуры

Методы:

+ Установить центр новой фигуры для умного создания

+ Сбросить центр для умного создания

+ Произвести умное создание

+ Начать быстрое создание

+ Закончить быстрое создание

+ Очистить изображение

+ Подключить доступные типы фигур

+ Перерисовать изображение

+ Запросить значение параметра (имя параметра : строка) : целочисл.

ГЛАВА 3. РЕАЛИЗАЦИЯ

Для решения поставленной задачи было написано приложение в среде Lazarus с использованием языка Delphi (см. скриншот приложения на рис. 11). Исходный код приведен в приложении А.

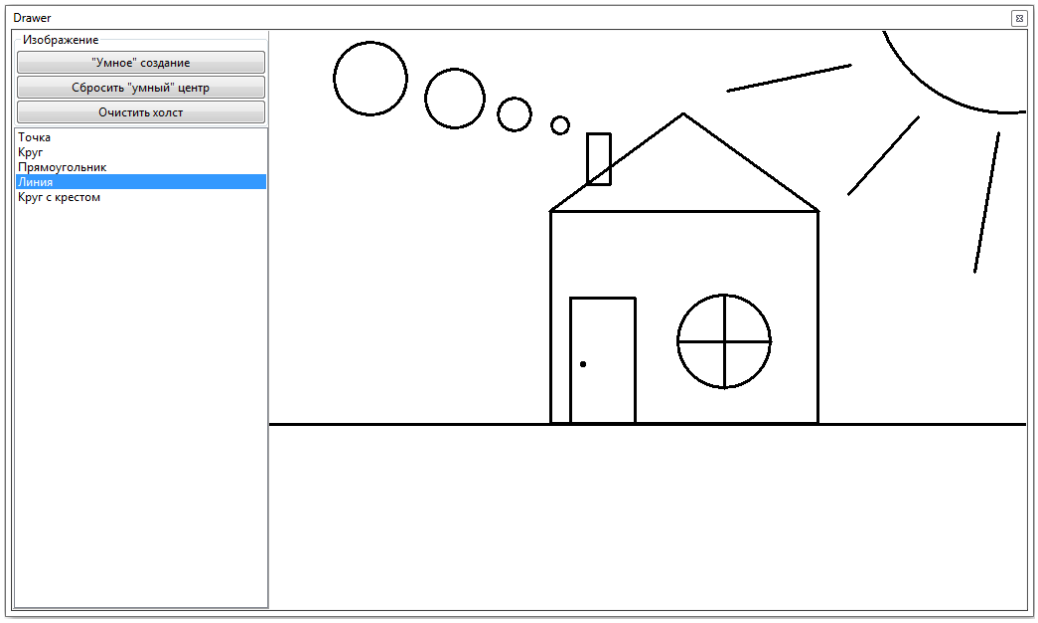

Рисунок 11 – Скриншот приложения

Таблица 1 демонстрирует реализованные модули и реализованные в них классы.

Таблица 1 – Модули приложения

Модуль	**Реализуемые классы**
uMainFormGUI	TMainForm
uAppSystem	AppSystem
uFigure	TFigure
uFigureTypes	TDot, TLine, TCircle, TRectangle, TCrossCircle
uFigureList	TFigureList, TListIterator
uArray	Tarray<T>

Ниже представлена диаграмма компонентов, которая показывает, каким образом собирается исполняемый файл (рис. 12).

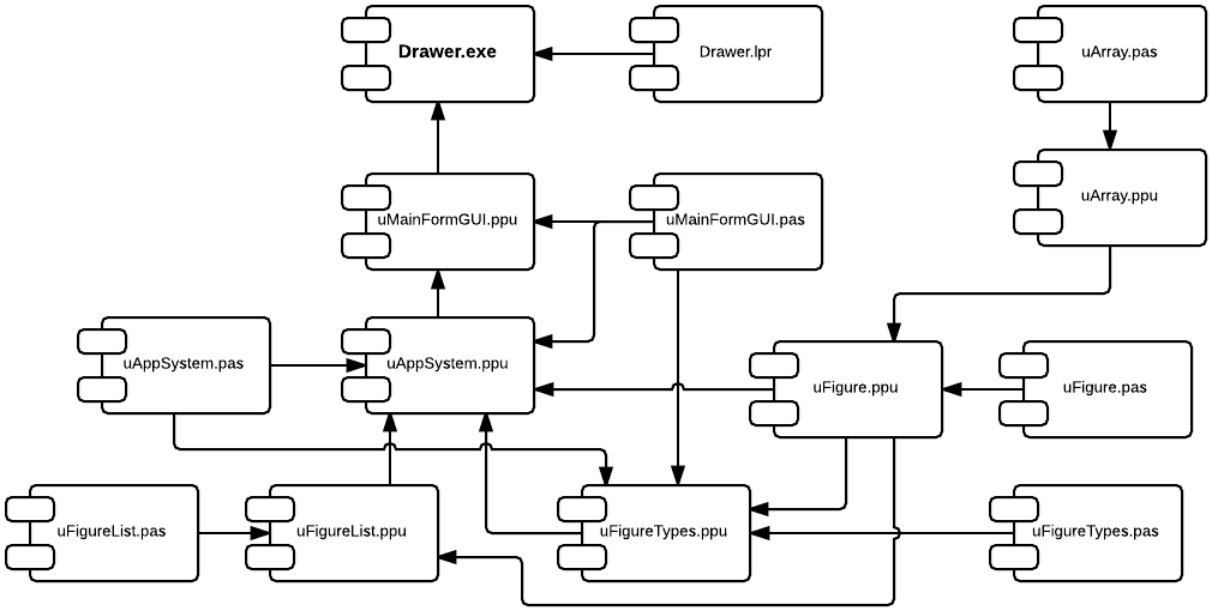

Рисунок 12 – Диаграмма компонентов

Таблица 2 описывает назначение компонентов.

Таблица 2 – Компоненты

Компонент	**Назначение**
Drawer.exe	Исполняемый файл приложения
Drawer.lpr	Исходный код программы
*.pas	Исходный код модуля с соответствующим названием
*.ppu	Откомпилированный модуль с соответствующим названием

На рисунке 13-15 приведено описание каждого созданного класса.

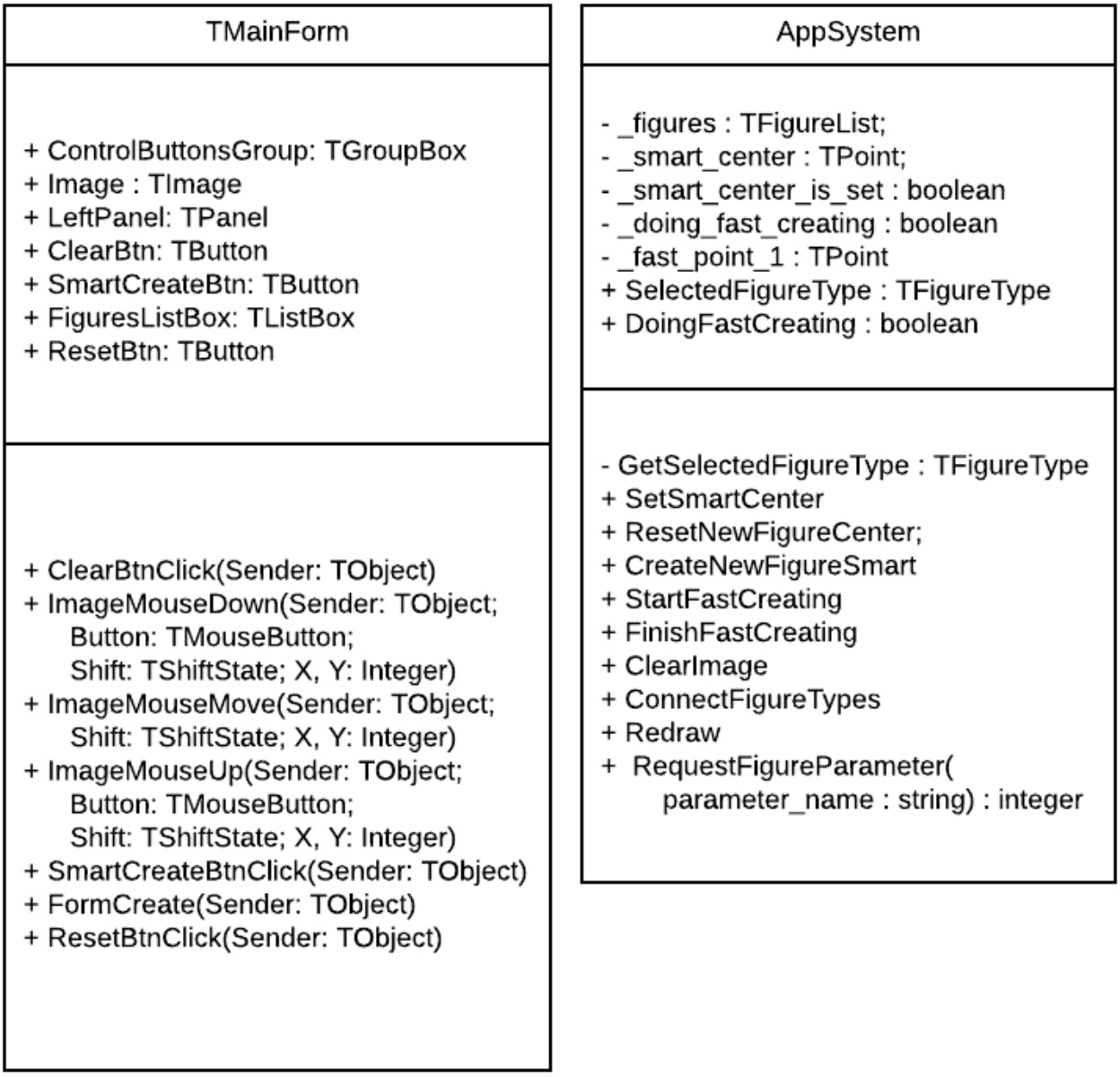

Рисунок 13 – Описание классов

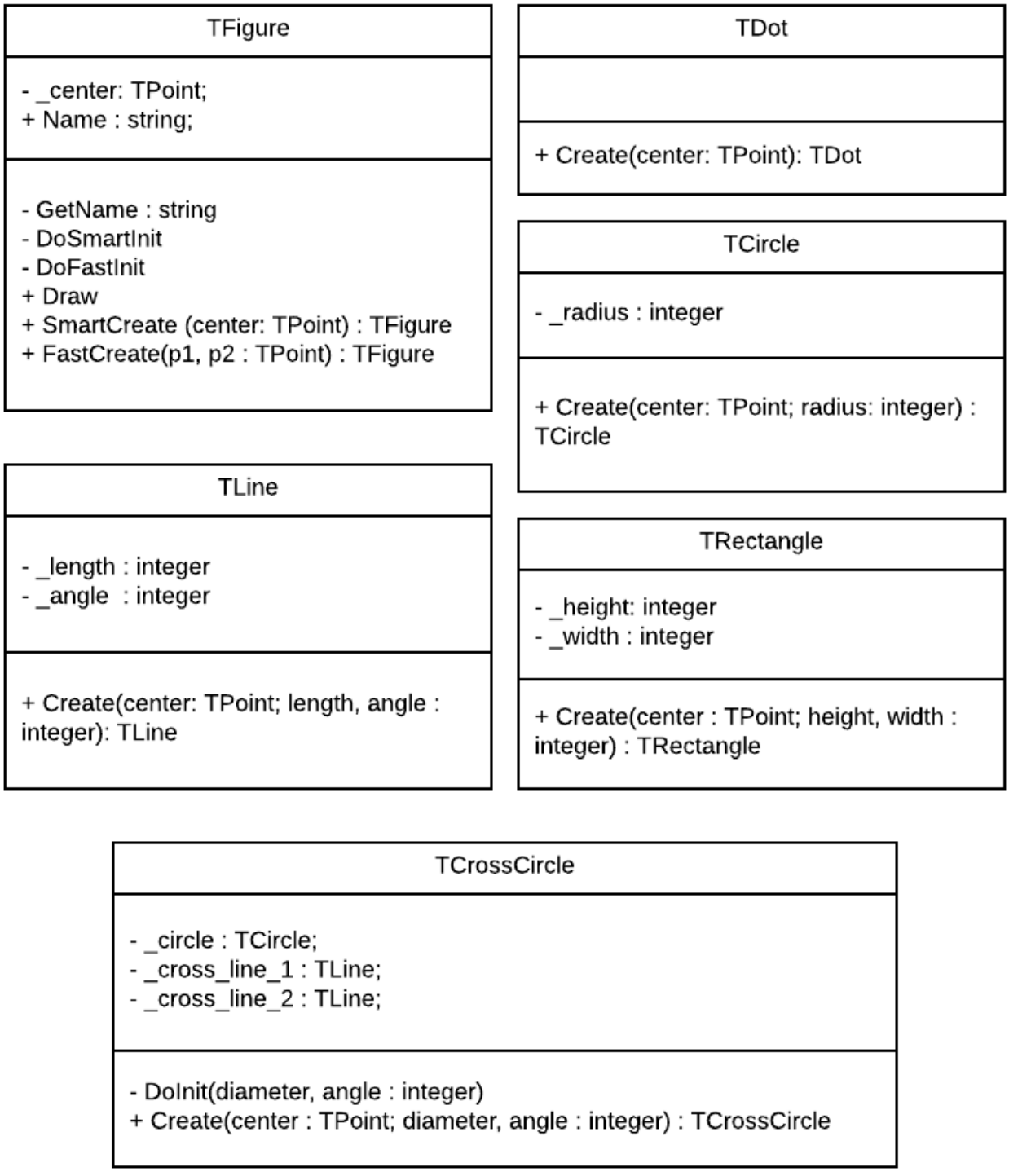

Рисунок 14 – Описание классов

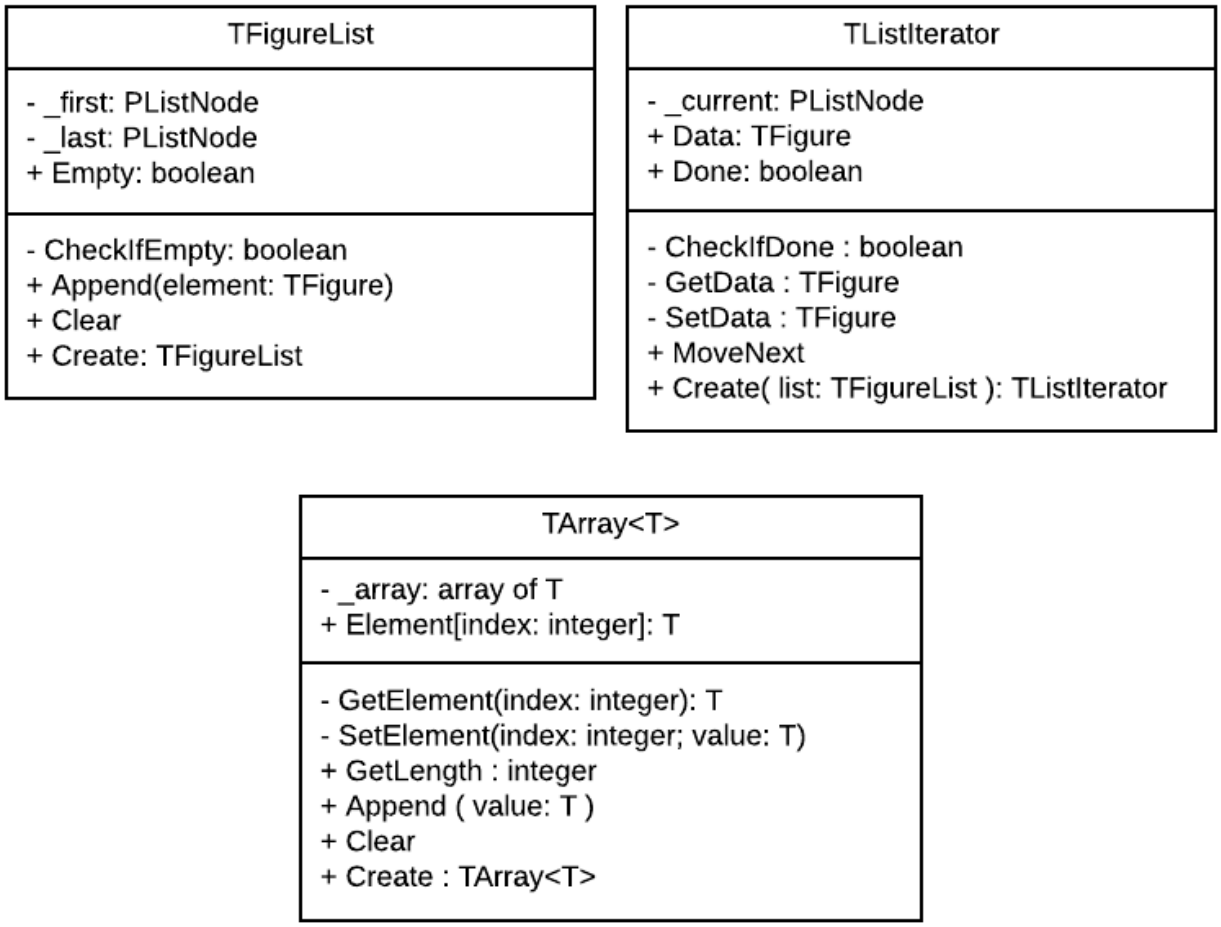

Рисунок 15 – Описание классов

ГЛАВА 4. ТЕСТИРОВАНИЕ

В таблице 3 представлены функциональные тесты

Таблица 3 – функциональные тесты

Вариант использования	**Параметры теста**	**Результат**
Выбрать фигуру	Точка	Пройден (рис. 22)
	Линия	Пройден (рис. 22)
	Круг	Пройден (рис. 22)
	Прямоугольник	Пройден (рис. 22)
	Круг с крестом	Пройден (рис. 22)
«Быстрое» создание	Круг с крестом	Пройден (рис. 16-17)
	Прямоугольник	Пройден
	Линия	Пройден
«Умное» создание	Прямоугольник с высотой 400 и шириной 200	Пройден (рис. 18-21)
	Круг с радиусом 50	Пройден
	Линия с длиной 200 и углом наклона 60	Пройден
Очистить изображение	Заполненное изображение	Пройден (рис. 23-24)
	Пустое изображение	Пройден

«Быстрое» создание

Цель теста: определить, позволяет ли приложение создавать фигуры, обозначая область, в которую они должны быть вписаны.

Ход выполнения:

1. В списке фигур выбрали необходимую фигуру.
2. Выделили прямоугольную область, в которую должна вписаться новая фигура.

Результат: на области рисования появилась выбранная фигура, которая вписалась в выделенную нами область.

Вывод: тест пройден.

«Умное» создание

Цель теста: определить, позволяет ли приложение создавать фигуры, явно указывая числовые значения их параметров.

Ход выполнения:

1. В списке фигур выбрали необходимую фигуру.
2. На области рисования обозначили точку, которая должна соответствовать центру новой фигуры.
3. Нажали на кнопку «Умное создание».
4. При каждом запросе параметра ввели необходимые значения.

Результат: на области рисования появилась выбранная фигура с заданными параметрами.

Вывод: тест пройден.

Выбрать фигуру

Цель теста: определить, позволяет ли приложение выбирать и рисовать фигуры, обозначенные в модели предметной области.

Ход выполнения:

1. Выбрали фигуру
2. Нарисовали выбранную фигуру

Результат: на области изображения появилась выбранная фигура.

Вывод: тест пройден.

Очистить изображение

Цель теста: определить, позволяет ли приложение очищать область рисования.

Ход выполнения:

1. Заполнили область рисования произвольным набором фигур.
2. Нажали на кнопку «Очистить холст»
3. Подтвердили действие, нажав кнопку «Yes».

Результат: изображение очистилось.

Вывод: тест пройден.

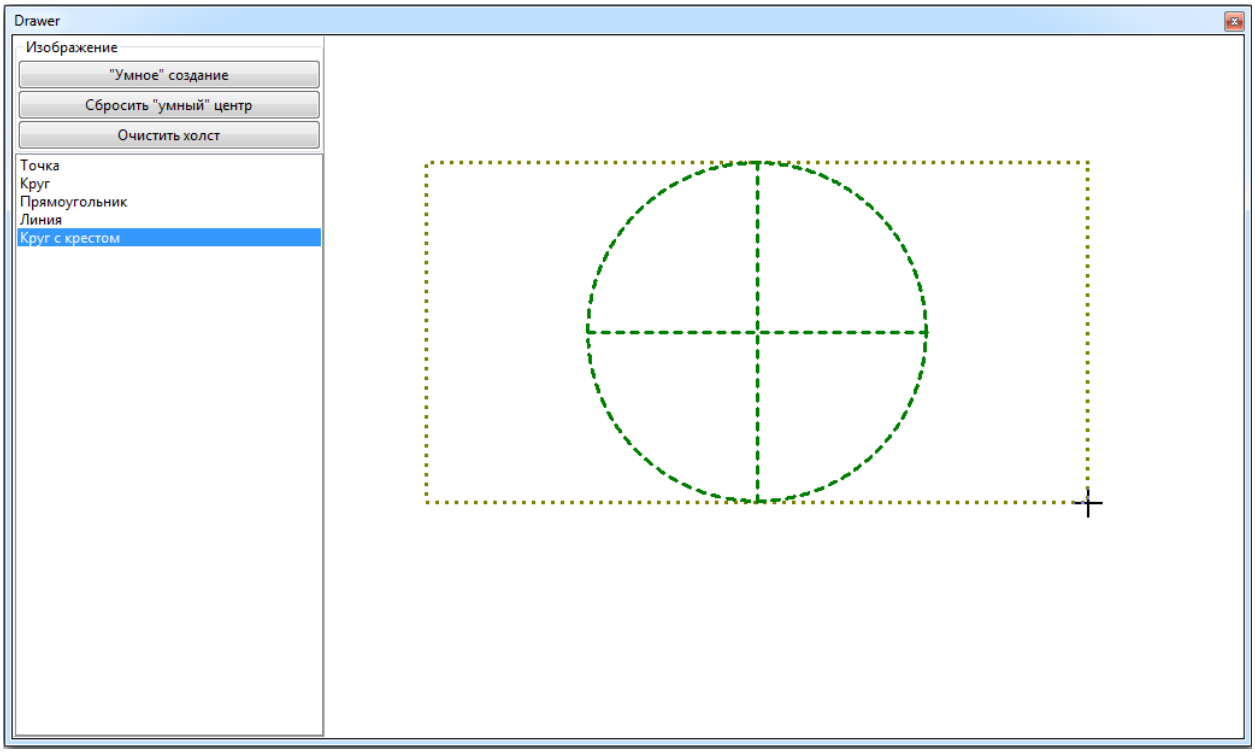

Рисунок 16 – Тест «быстрого» создания

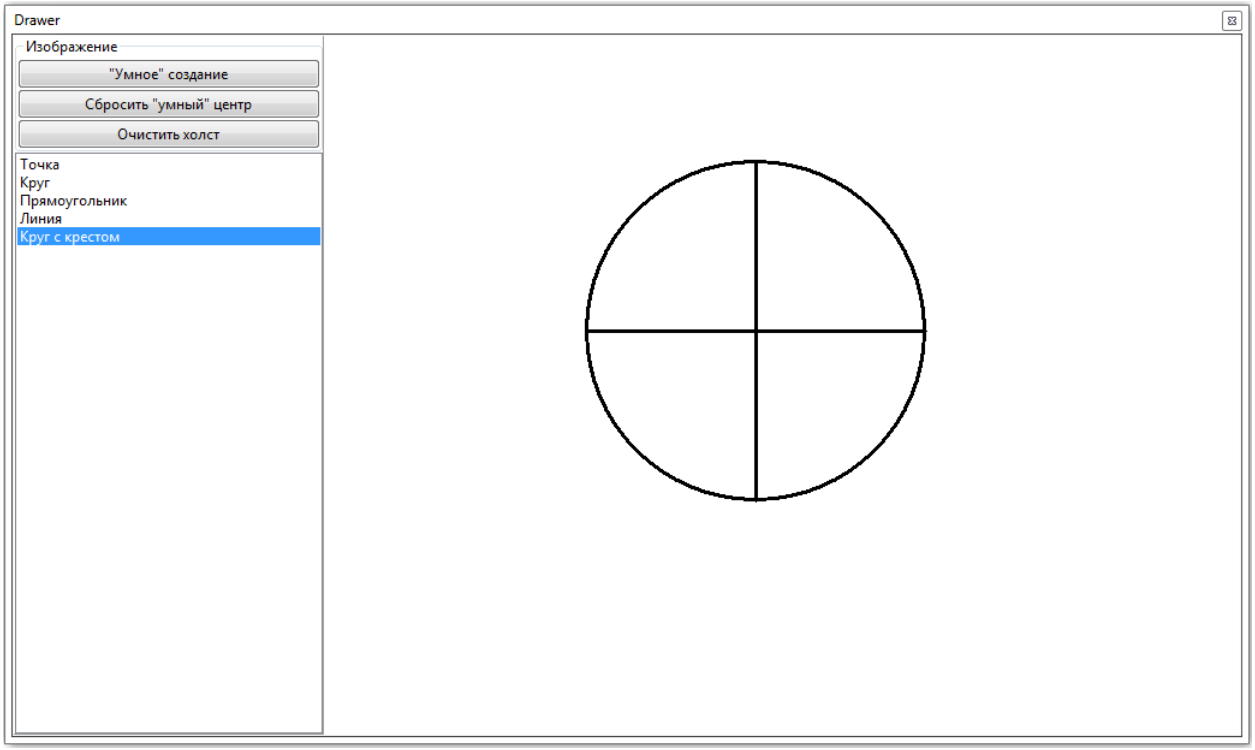

Рисунок 17 – Тест «быстрого» создания

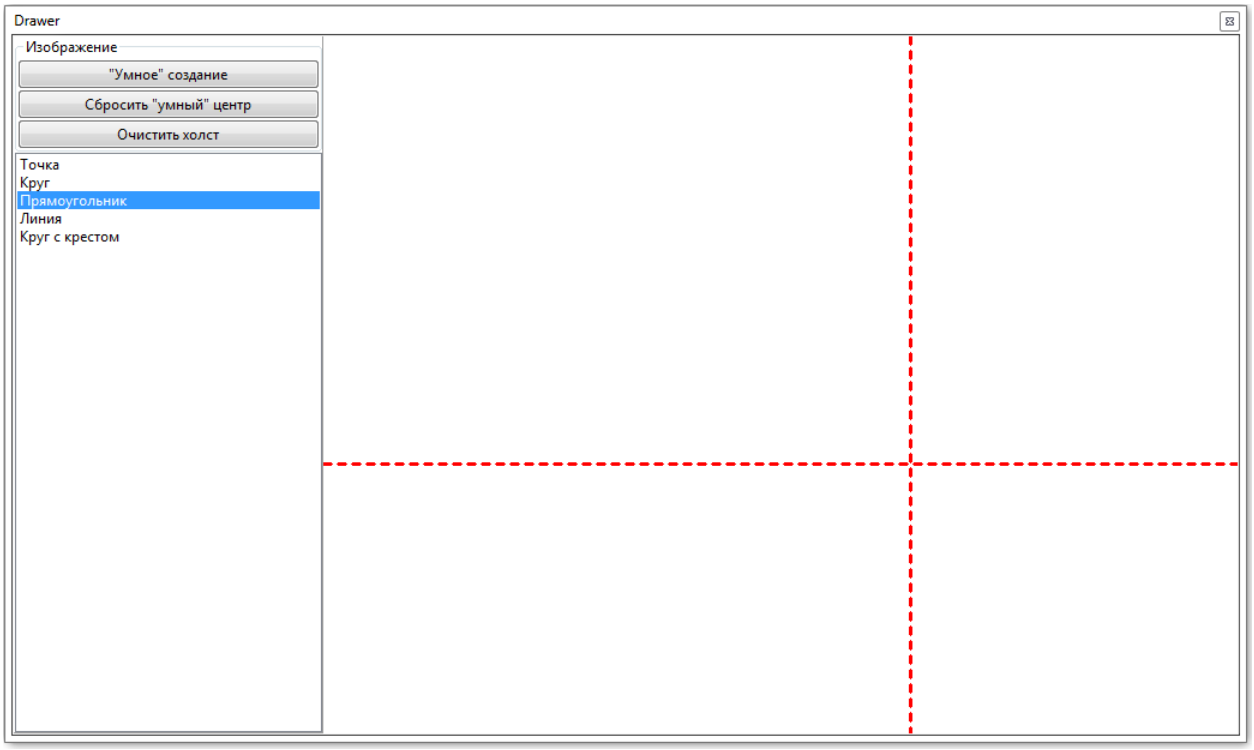

Рисунок 18 – Тест «умного» создания

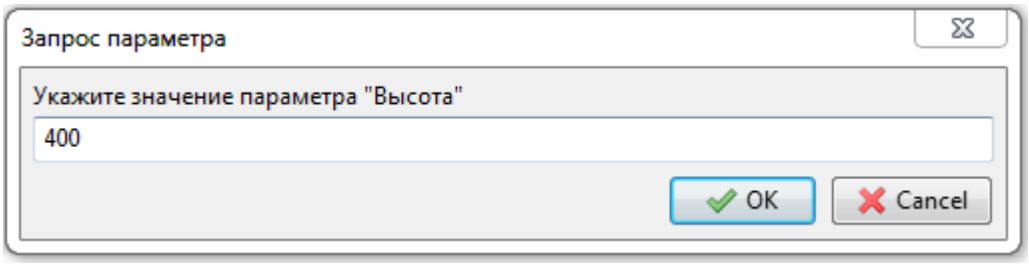

Рисунок 19 – Тест «умного» создания

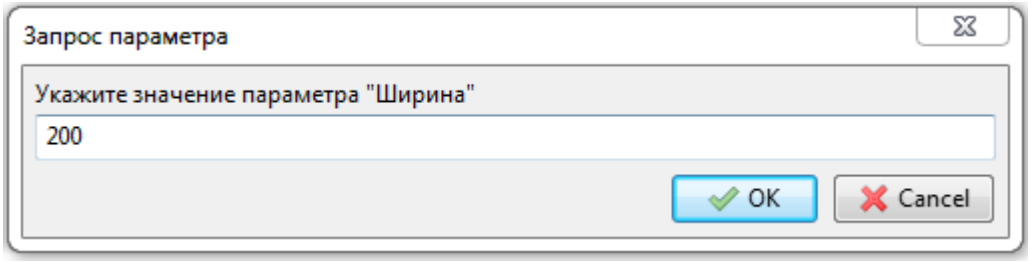

Рисунок 20 – Тест «умного» создания

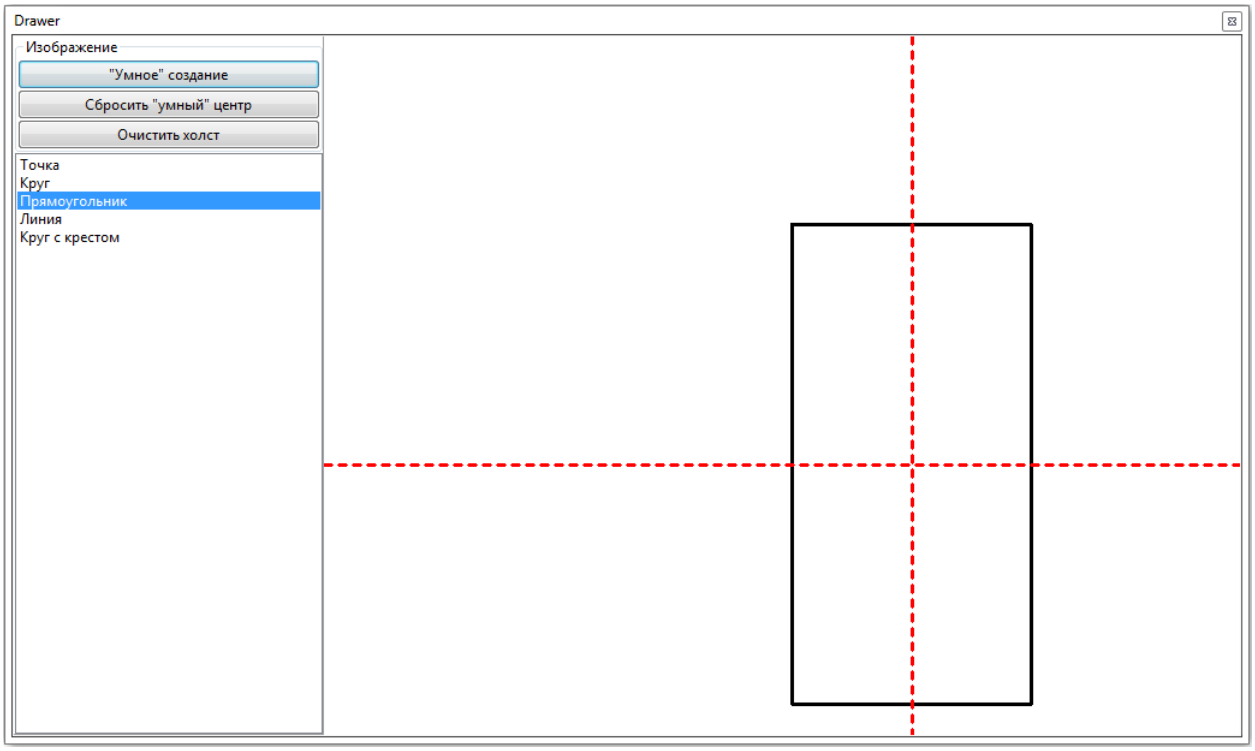

Рисунок 21 – Тест «умного» создания

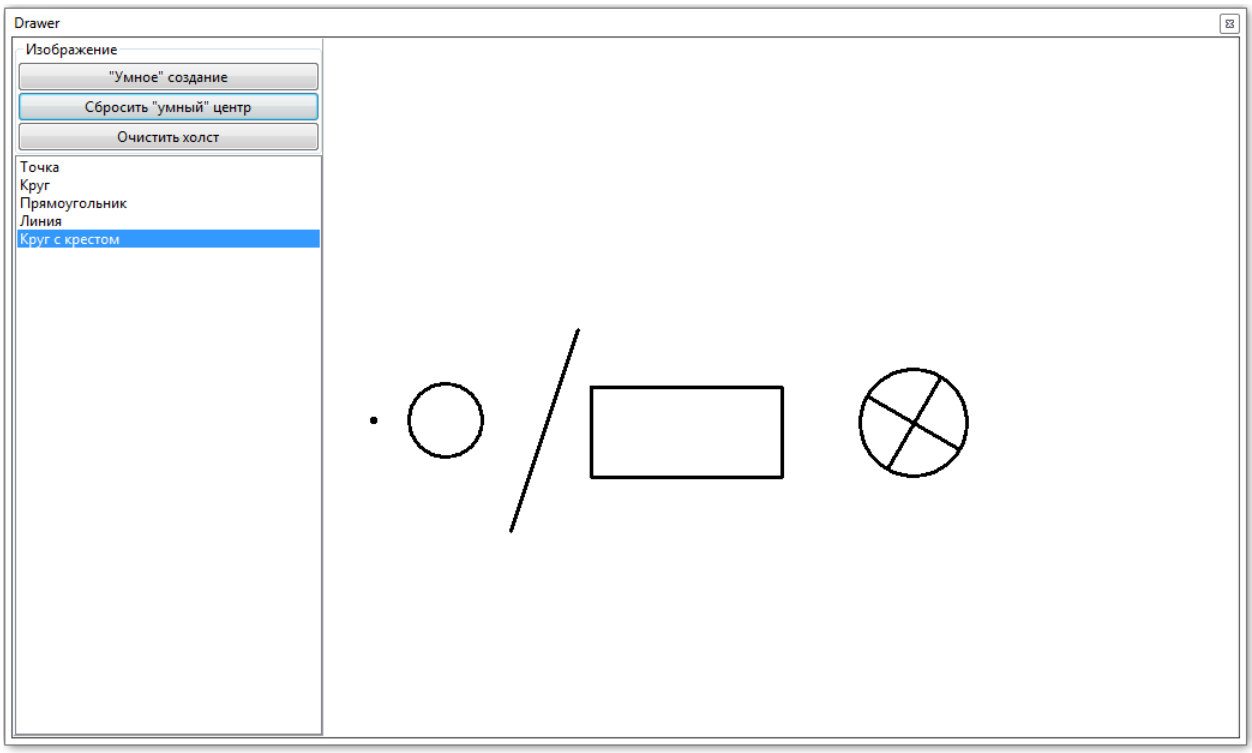

Рисунок 22 – Тест допустимых типов фигур.

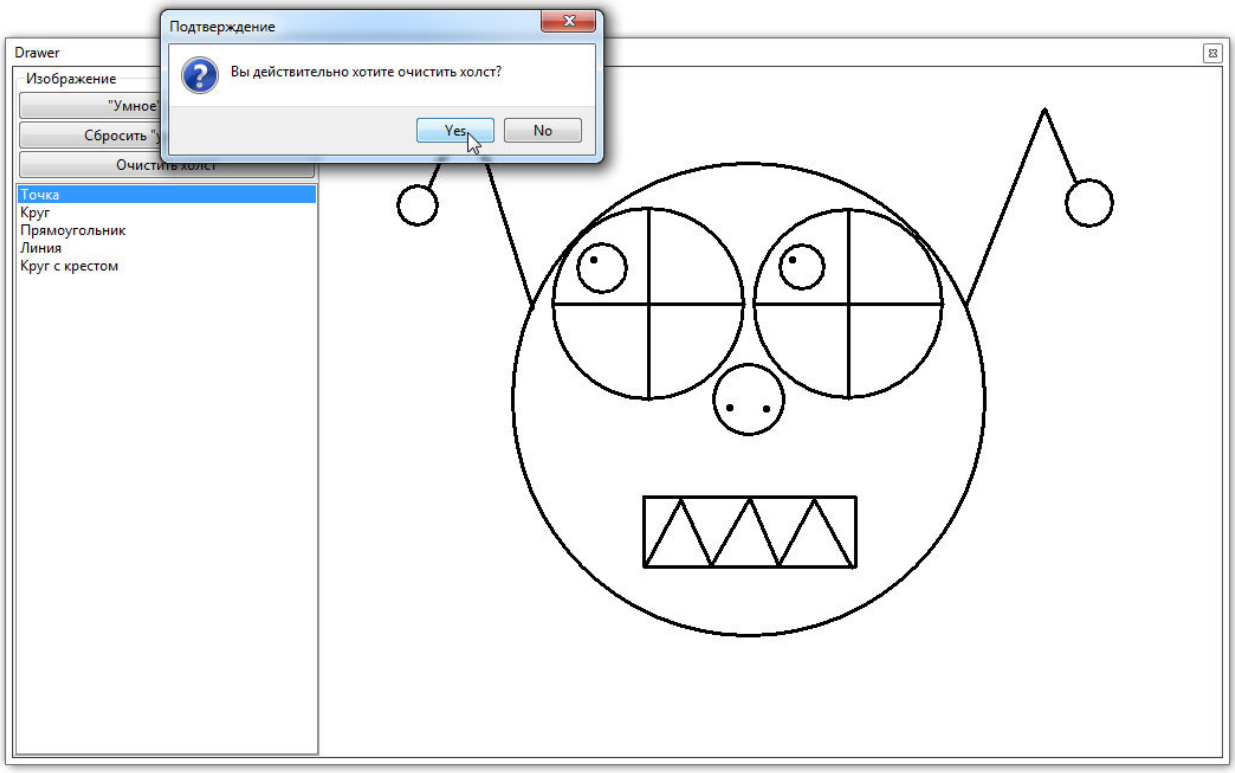

Рисунок 23 – Тест очистки изображения

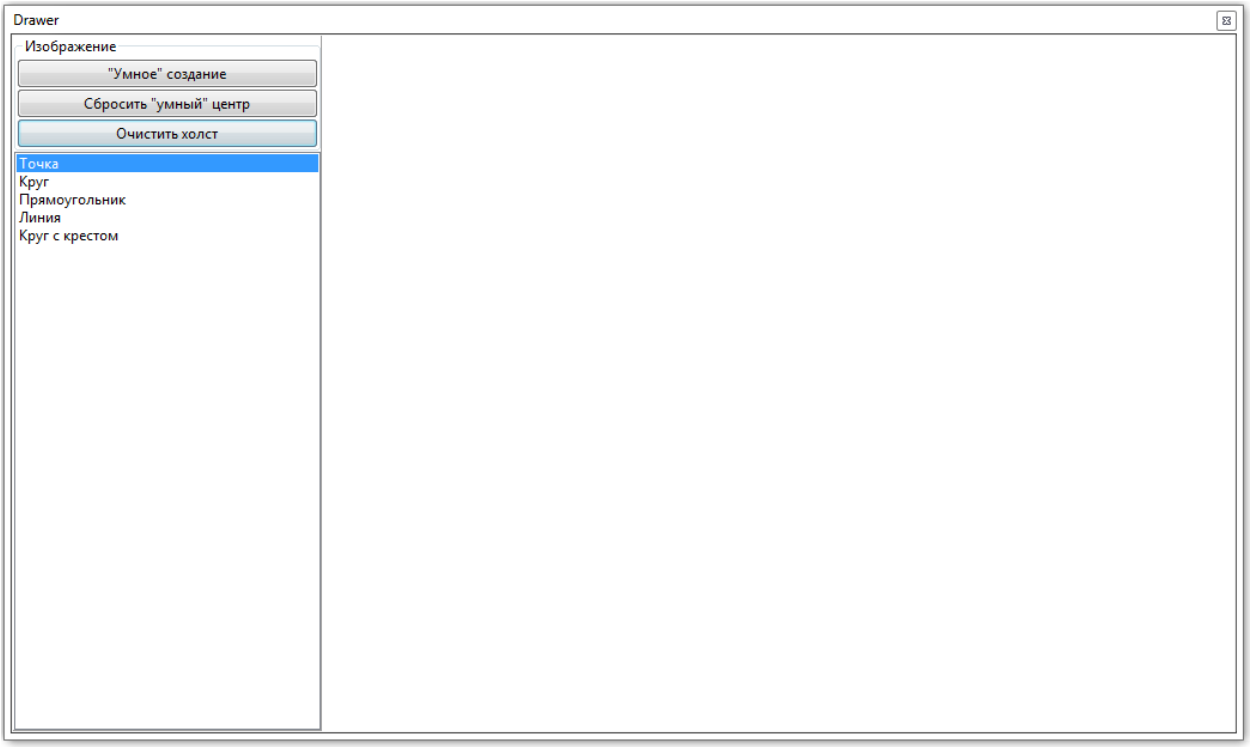

Рисунок 24 – Тест очистки изображения

ЗАКЛЮЧЕНИЕ

Результатом работы является приложение «Drawer», написанное на языке Delphi в среде Lazarus. Данное приложение удовлетворяет всем изложенным требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гурьянов Л.В., Гурьянова Л.С., Мещеряков Б.К., Ракова А.Н. / Основы конструирования программ. Лабораторный практикум. – Пенза, Изд-во ПГУ, 2010
2. Информационный портал для разработчиков на Free Pascal / www.freepascal.ru/
3. Lazarus для школьников и студентов / sites.google.com/site/studylazarus/
4. Программирование на Lazarus / intuit.valrkl.ru/course-1265/index.html

ПРИЛОЖЕНИЕ А – ЛИСТИНГ ИСХОДНОГО КОДА

Листинг модуля «uMainFormGUI.pas»

```
unit uMainFormGUI;

{$mode delphi}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
  ExtCtrls, Menus, ValEdit, ComCtrls, ActnList, DbCtrls, ColorBox, Grids,
  // Модули проекта
  uAppSystem;

type

  { TMainForm }

  TMainForm = class(TForm)
    ControlButtonsGroup: TGroupBox;
    Image : TImage;
    LeftPanel: TPanel;
    ClearBtn: TButton;
    SmartCreateBtn: TButton;
    FiguresListBox: TListBox;
    ResetBtn: TButton;
    procedure ClearBtnClick(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 SmartCreateBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ResetBtnClick(Sender: TObject);
  end;

var
  MainForm: TMainForm;

implementation

{$R *.lfm}

// Кнопки
// -------------------------------------------------------

procedure TMainForm.SmartCreateBtnClick(Sender: TObject);
begin
  AppSystem.CreateNewFigureSmart;
  AppSystem.Redraw;
end;

procedure TMainForm.ResetBtnClick(Sender: TObject);
begin
```

```
  AppSystem.ResetNewFigureCenter;
  AppSystem.Redraw;
end;

procedure TMainForm.ClearBtnClick(Sender: TObject);
begin
  AppSystem.ClearImage;
  AppSystem.Redraw;
end;

// Изображение
// -----------------------------------------------------

procedure TMainForm.ImageMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
    AppSystem.StartFastCreating;
end;

procedure TMainForm.ImageMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  if AppSystem.DoingFastCreating then
    AppSystem.Redraw;
end;

procedure TMainForm.ImageMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
  begin
    AppSystem.FinishFastCreating;
    AppSystem.Redraw;
  end
  else if Button = mbRight then
  begin
    AppSystem.SetSmartCenter;
    AppSystem.Redraw;
  end;
end;

// Форма
// -----------------------------------------------------

procedure TMainForm.FormCreate(Sender: TObject);
begin
  AppSystem.ConnectFigureTypes;
end;

end.
```

Листинг модуля «uAppSystem.pas»

```
unit uAppSystem;

{$mode delphi}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
  ExtCtrls, Menus, ValEdit, ComCtrls,
  // Модули проекта
  uFigure, uFigureList, uFigureTypes;

type

  AppSystem = class // Статический класс
  private
    // Поля
    class var _figures             : TFigureList;
    class var _smart_center        : TPoint;
    class var _smart_center_is_set : boolean;
    class var _doing_fast_creating : boolean;
    class var _fast_point_1        : TPoint;
    // Акцессоры
    class function GetSelectedFigureType : TFigureType; static;
  public
    // Свойства
    class property SelectedFigureType : TFigureType
      read GetSelectedFigureType;
    class property DoingFastCreating: boolean
      read _doing_fast_creating;
    // Процедуры
    class procedure SetSmartCenter;       static;
    class procedure ResetNewFigureCenter; static;
    class procedure CreateNewFigureSmart; static;
    class procedure StartFastCreating;    static;
    class procedure FinishFastCreating;   static;
    class procedure ClearImage;           static;
    class procedure ConnectFigureTypes;   static;
    class procedure Redraw;               static;
    // Функции
    class function  RequestFigureParameter
                    ( const parameter_name : string ) : integer; static;
  end;

  ERequestDeniedByUser = class(Exception)
  end;

implementation

uses
  uMainFormGUI;

class procedure AppSystem.SetSmartCenter;
begin
  with MainForm do
  begin
    _smart_center := Image.ScreenToClient(mouse.CursorPos);
```

```
    _smart_center_is_set := true;
  end;
end;

class procedure AppSystem.ResetNewFigureCenter;
begin
  _smart_center_is_set := false;
end;

class procedure AppSystem.CreateNewFigureSmart;
var
  figure : TFigure;
begin
  if _smart_center_is_set then
  begin
    try
      figure := SelectedFigureType.SmartCreate( _smart_center );
    except
      on ERequestDeniedByUser do exit;
    end;
    _figures.Append( figure )
  end else
    ShowMessage('Щелчком правой кнопки мыши укажите центр новой фигуры');
end;

class procedure AppSystem.StartFastCreating;
begin
  _fast_point_1 := MainForm.Image.ScreenToClient(mouse.CursorPos);
  _doing_fast_creating := true;
end;

class procedure AppSystem.FinishFastCreating;
var
  figure : TFigure;
  p2     : TPoint;
begin
  if _doing_fast_creating then
  begin
    p2 := MainForm.Image.ScreenToClient(mouse.CursorPos);;
    figure := SelectedFigureType.FastCreate( _fast_point_1, p2 );
    _figures.Append( figure );
  end;
  _doing_fast_creating := false;
end;

class function AppSystem.GetSelectedFigureType : TFigureType;
begin
  result := FigureTypes[ MainForm.FiguresListBox.ItemIndex ];
end;

class procedure AppSystem.Redraw;
const
  // "Pen Width" = "pw"
  pw_EXISTING_FIGURE    = 3;
  pw_SMART_CENTER_CROSS = 3;
  pw_FAST_BOUNDS        = 4;
  pw_FAST_SILHOUETTE    = 3;
var
  it         : TListIterator;
  cursor_pos : TPoint;
```

```
begin
  with MainForm.Image.Canvas do
  begin
    Lock;   // Холст заблокировани
    // ==================================
    // Очистка холста
    Brush.Style := bsSolid;
    FillRect( ClipRect );

    // Параметры контура и заливки фигур
    Pen.Color   := clBlack;
    Pen.Width   := pw_EXISTING_FIGURE;
    Pen.Style   := psSolid;
    Brush.Style := bsClear;

    // Прорисовка фигур из списка
    it := TListIterator.Create(_figures);
    while not it.Done do
    begin
      it.Data.Draw;
      it.MoveNext;
    end; // while

    // Пометка центра новой фигуры
    if _smart_center_is_set then
    begin
      Pen.Color := clRed;
      Pen.Width := pw_SMART_CENTER_CROSS;
      Pen.Style := psDash;
      Line( 0,                    _smart_center.y,
            MainForm.Image.Width, _smart_center.y );
      Line( _smart_center.x,    0,
            _smart_center.x,    MainForm.Image.Height );
    end; // if

    // Пометка области фигуры при быстром создания
    cursor_pos := MainForm.Image.ScreenToClient(mouse.CursorPos);
    if _doing_fast_creating then
    begin
      Pen.Color := clOlive;
      Pen.Width := pw_FAST_BOUNDS;
      Pen.Style := psDot;
      Rectangle( _fast_point_1.x, _fast_point_1.y,
                 cursor_pos.x,
                 cursor_pos.y);
      // Силуэт новой фигуры при быстром создании
      Pen.Color := clGreen;
      Pen.Width := pw_FAST_SILHOUETTE;
      Pen.Style := psDash;
      (
        SelectedFigureType.FastCreate( _fast_point_1, cursor_pos )
      ).Draw;
    end; // if
    // ==================================
    Unlock; // Холст разблокирован
  end; // with
end;

class procedure AppSystem.ClearImage;
var
```

```
  answer : TModalResult;
begin
  if not _figures.Empty then
  begin
    answer := MessageDlg( 'Подтверждение',
                          'Вы действительно хотите очистить холст?',
                          mtConfirmation, mbYesNo, 0);
    if answer = mrYes then
      _figures.Clear;
  end;
end;

class procedure AppSystem.ConnectFigureTypes;
var
  i : integer;
  figure_type : TFigureType;
begin
  for i := 0 to FigureTypes.GetLength - 1 do
  begin
    figure_type := FigureTypes[i];
    MainForm.FiguresListBox.AddItem( figure_type.Name, nil );
  end;
  MainForm.FiguresListBox.ItemIndex := 0;
end;

class function AppSystem.RequestFigureParameter
                              (const parameter_name: string): integer;
const
  DEFAULT_VALUE = 50;
var
  input_implemented : boolean;
  input_expression : string;
begin
  input_implemented :=
  InputQuery(
        'Запрос параметра',
        'Укажите значение параметра "' + parameter_name + '"',
  input_expression);
  if input_implemented then
    try
      result := StrToInt(input_expression);
    except
      ShowMessage( 'Ошибка ввода. Используется значение по умолчанию' );
      result := DEFAULT_VALUE;
    end
  else
    raise ERequestDeniedByUser.Create('Запрос параметра отклонен');
end;

initialization

  AppSystem._figures := TFigureList.Create;
  AppSystem._smart_center_is_set := false;
  AppSystem._doing_fast_creating := false;

finalization

  AppSystem._figures.Free;

end.
```

Листинг модуля «uFigure.pas»

```
unit uFigure;

{$mode delphi}{$H+}

interface

uses
  Classes, SysUtils,
  // Модули проекта
  uArray;

type

  TFigureType = class of TFigure;

  TFigure = class
  protected
    _center : TPoint;
    procedure DoSmartInit;                          virtual; abstract;
    procedure DoFastInit( const p1, p2 : TPoint ); virtual; abstract;
    class function GetName : string; virtual; abstract; static;
  public
    procedure Draw; virtual; abstract;
    class property Name : string
      read GetName;
    constructor SmartCreate( const center : TPoint );
    constructor FastCreate ( const p1,  p2 : TPoint );
  end;

// Глобальный контейнер типов фигур
// ------------------------------------
var FigureTypes : TArray<TFigureType>;

implementation

constructor TFigure.SmartCreate( const center : TPoint );
begin
  _center := center;
  DoSmartInit;
end;

constructor TFigure.FastCreate( const  p1, p2 : TPoint );
begin
  _center.x := p1.x + (p2.x - p1.x) div 2;
  _center.y := p1.y + (p2.y - p1.y) div 2;
  DoFastInit( p1, p2 );
end;

initialization

  FigureTypes := TArray<TFigureType>.Create;

finalization

  FigureTypes.Free;

end.
```

Листинг модуля «uFigureTypes.pas»

```
unit uFigureTypes;

interface

uses
  Classes, SysUtils, Forms, Graphics,
  // Модули проекта
  uFigure;

type

// Описание конкретных классов фигур
// =============================================
  TDot = class( TFigure )
  protected
    procedure DoSmartInit;                          override;
    procedure DoFastInit( const p1, p2 : TPoint ); override;
    class function GetName : string; override; static;
  public
    procedure Draw; override;
    constructor Create( const center : TPoint );
  end;
// ---------------------------------------------
  TCircle = class( TFigure )
  protected
    _radius : integer;
    procedure DoSmartInit;                          override;
    procedure DoFastInit( const p1, p2 : TPoint ); override;
    class function GetName : string; override; static;
  public
    procedure Draw; override;
    constructor Create( const center : TPoint; radius: integer );
  end;
// ---------------------------------------------
  TRectangle = class( TFigure )
  protected
    _height : integer;
    _width  : integer;
    procedure DoSmartInit;                          override;
    procedure DoFastInit( const p1, p2 : TPoint ); override;
    class function GetName : string; override; static;
  public
    procedure Draw; override;
    constructor Create( const center : TPoint;
                        height, width : integer );
  end;
// ---------------------------------------------
  TLine = class( TFigure )
  protected
    _length : integer;
    _angle  : integer;
    procedure DoSmartInit;                          override;
    procedure DoFastInit( const p1, p2 : TPoint ); override;
    class function GetName : string; override; static;
  public
    procedure Draw; override;
```

```
    constructor Create( const center : TPoint;
                        length, angle : integer );
  end;
// -----------------------------------------------
  TCrossCircle = class( TFigure )
  protected
    _circle       : TCircle;
    _cross_line_1 : TLine;
    _cross_line_2 : TLine;
    procedure DoInit( diameter, angle : integer );
    procedure DoSmartInit;                          override;
    procedure DoFastInit( const p1, p2 : TPoint ); override;
    class function GetName : string; override; static;
  public
    procedure Draw; override;
    constructor Create( const center : TPoint;
                        diameter, angle : integer );
    destructor  Destroy; override;
  end;

// Конец описания
// ==============================================

implementation

uses
  uMainFormGUI, uAppSystem;

// TDot
// -----------------------------------------------
class function TDot.GetName : string;
begin
  result := 'Точка';
end;

procedure TDot.DoSmartInit; begin end;

procedure TDot.DoFastInit( const p1, p2 : TPoint ); begin end;

procedure TDot.Draw;
begin
  MainForm.Image.Canvas.Ellipse( _center.x - 2, _center.y - 2,
                                 _center.x + 2, _center.y + 2);
end;

constructor TDot.Create( const center : TPoint );
begin
  _center := center;
end;

// TCircle
// -----------------------------------------------
class function TCircle.GetName : string;
begin
  result := 'Круг';
end;

procedure TCircle.DoSmartInit;
begin
  _radius := AppSystem.RequestFigureParameter('Радиус');
```

```
end;

procedure TCircle.DoFastInit( const p1, p2 : TPoint );
var
  height : integer;
  width  : integer;
begin
  height  := Abs( p2.y - p1.y );
  width   := Abs( p2.x - p1.x );
  if height < width then
    _radius := height div 2
  else
    _radius := width  div 2;
end;

procedure TCircle.Draw;
begin
  MainForm.Image.Canvas.Ellipse(_center.x - _radius,
                                _center.y - _radius,
                                _center.x + _radius,
                                _center.y + _radius );
end;

constructor TCircle.Create( const center : TPoint; radius: integer );
begin
  _center := center;
  _radius := radius;
end;

// TRectangle
// -----------------------------------------
class function TRectangle.GetName : string;
begin
  result := 'Прямоугольник';
end;

procedure TRectangle.DoSmartInit;
begin
  _height := AppSystem.RequestFigureParameter('Высота');
  _width  := AppSystem.RequestFigureParameter('Ширина');
end;

procedure TRectangle.DoFastInit( const p1, p2 : TPoint );
begin
  _height  := Abs( p2.y - p1.y );
  _width   := Abs( p2.x - p1.x );
end;

procedure TRectangle.Draw;
var
  half_height : integer;
  half_width  : integer;
begin
  half_height := _height div 2;
  half_width  := _width  div 2;
  MainForm.Image.Canvas.Rectangle(_center.x - half_width,
                                  _center.y - half_height,
                                  _center.x + half_width,
                                  _center.y + half_height );
end;
```

```
constructor TRectangle.Create(const center: TPoint;
                              height, width: integer);
begin
  _center := center;
  _height := height;
  _width  := width;
end;

// TLine
// -----------------------------------------------
class function TLine.GetName : string;
begin
  result := 'Линия';
end;

procedure TLine.DoSmartInit;
begin
  _length := AppSystem.RequestFigureParameter('Длина');
  _angle  := AppSystem.RequestFigureParameter('Угол наклона');
end;

procedure TLine.DoFastInit( const p1, p2 : TPoint );
var
  dx : integer;
  dy : integer;
begin
  dx := p2.x - p1.x;
  dy := p1.y - p2.y;
  if dx = 0 then
  begin
    _angle := 90
  end else
  begin
    _angle := trunc( arctan(dy / dx) * 180 / pi );
  end;
  _length  := trunc( sqrt( sqr(dy) + sqr(dx) )  );
end;

procedure TLine.Draw;
var
  radians     : real;
  half_length : integer;
  edge        : TPoint;
begin
  radians     :=  -_angle * pi / 180 ;
  half_length :=  _length div 2;

  edge.x := Trunc( cos(radians) * half_length );
  edge.y := Trunc( sin(radians) * half_length );

  MainForm.Image.Canvas.Line( edge.x + _center.x,  edge.y + _center.y,
                             -edge.x + _center.x, -edge.y + _center.y);
end;

constructor TLine.Create( const center : TPoint;
                          length, angle : integer );
begin
  _center := center;
  _length := length;
```

```
  _angle  := angle;
end;

// TCrossCircle
// -----------------------------------------
class function TCrossCircle.GetName : string;
begin
  result := 'Круг с крестом';
end;

procedure TCrossCircle.DoInit(diameter, angle : integer );
begin
  _circle       := TCircle.Create( _center, diameter div 2       );
  _cross_line_1 := TLine.Create  ( _center, diameter, angle      );
  _cross_line_2 := TLine.Create  ( _center, diameter, angle + 90 );
end;

procedure TCrossCircle.DoSmartInit;
var
  diameter : integer;
  angle    : integer;
begin
  diameter := AppSystem.RequestFigureParameter('Диаметр');
  angle    := AppSystem.RequestFigureParameter('Угол наклона');
  DoInit(diameter, angle);
end;

procedure TCrossCircle.DoFastInit( const p1, p2 : TPoint );
const
  DEFAULT_ANGLE = 0;
var
  diameter : integer;
  height   : integer;
  width    : integer;
begin
  height  := Abs( p2.y - p1.y );
  width   := Abs( p2.x - p1.x );
  if height < width then
    diameter := height
  else
    diameter := width;
  DoInit(diameter, DEFAULT_ANGLE);
end;

procedure TCrossCircle.Draw;
begin
  _circle.Draw;
  _cross_line_1.Draw;
  _cross_line_2.Draw;
end;

constructor TCrossCircle.Create( const center : TPoint;
                                 diameter, angle : integer);
begin
  _center := center;
  DoInit(diameter, angle);
end;

destructor TCrossCircle.Destroy;
begin
```

```
  _circle.Free;
  _cross_line_1.Free;
  _cross_line_2.Free;
  inherited;
end;

initialization

  FigureTypes.Append( TDot );
  FigureTypes.Append( TCircle );
  FigureTypes.Append( TRectangle );
  FigureTypes.Append( TLine );
  FigureTypes.Append( TCrossCircle );

end.
```

Листинг модуля «uFigureList.pas»

```
unit uFigureList;

interface

uses
  uFigure;

type

  PListNode = ^TListNode;
  TListNode = record
    data : TFigure;
    next : PListNode;
  end;

  TFigureList = class
  private
    _first   : PListNode;
    _last    : PListNode;
    function CheckIfEmpty : boolean;
  public
    property Empty : boolean
      read CheckIfEmpty;
    procedure Append(const element: TFigure );
    procedure Clear;
    constructor Create;
    destructor  Destroy; override;
  end;

  TListIterator = class
  private
    _current : PListNode;
    function  GetData : TFigure;
    procedure SetData( const value : TFigure);
    function  CheckIfDone : boolean;
  public
    property Data : TFigure
      read  GetData
      write SetData;
    property Done : boolean
      read CheckIfDone;
    procedure   MoveNext;
    constructor Create( list: TFigureList );
  end;


implementation

// TFigureList
// =============================================
constructor TFigureList.Create;
begin
  _first := nil;
  _last  := nil;
end;
```

```
destructor TFigureList.Destroy;
begin
  Clear;
  inherited;
end;

function TFigureList.CheckIfEmpty: Boolean;
begin
  Result := _first = nil;
end;

procedure TFigureList.Append(const element: TFigure);
var
  new_node: PListNode;
begin
  if Empty then
  begin
    New(_first);
    _first^.data := element;
    _first^.next := nil;
    _last := _first;
  end else
  begin
    New(new_node);
    new_node^.data := element;
    new_node^.next := nil;
    _last^.next := new_node;
    _last := new_node;
  end;
end;

procedure TFigureList.Clear;
var
  current_node, next_node: PListNode;
begin
  if not Empty then
  begin
    current_node := _first;
    _first := nil;
    _last  := nil;
    while current_node <> nil do
    begin
      next_node := current_node^.next;
      current_node^.data.Free;
      Dispose(current_node);
      current_node := next_node;
    end;
  end;
end;

// TListIterator
// =============================================

function TListIterator.GetData: TFigure;
begin
  result := _current^.data;
end;

procedure TListIterator.SetData( const value : TFigure);
begin
```

```
  _current.Data := value;
end;

procedure TListIterator.MoveNext;
begin
  if _current <> nil then
    _current := _current^.next;
end;

function TListIterator.CheckIfDone: boolean;
begin
  result := _current = nil;
end;

constructor TListIterator.Create(list: TFigureList);
begin
  _current := list._first;
end;

end.
```

Листинг модуля «uArray.pas»

```
unit uArray;

interface

type
  TArray<T> = class
  private
    _array: array of T;
    function  GetElement(index: integer): T;
    procedure SetElement(index: integer; value: T);
  public
    property  Element[index: integer]: T
      read  GetElement
      write SetElement; default;
    function GetLength : integer;
    procedure Append ( value: T );
    procedure Clear;
    constructor Create;
    destructor  Destroy; override;
  end;

implementation

  function TArray<T>.GetElement(index: integer): T;
  begin
    result := _array[index];
  end;
  procedure TArray<T>.SetElement(index: integer; value: T);
  begin
    if index < Length(_array) then
      _array[index] := value;
  end;

  function TArray<T>.GetLength: integer;
  begin
    result := Length(_array);
  end;

  procedure TArray<T>.Append(value: T);
  begin
    SetLength( _array, Length(_array)+1 );
    _array[ High(_array) ] := value;
  end;
  procedure TArray<T>.Clear;
  begin
    SetLength( _array, 0 );
  end;
  constructor TArray<T>.Create;
  begin
    _array := nil;
  end;
  destructor TArray<T>.Destroy;
  begin
    Clear;
    inherited;
  end;
end.
```