

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Пензенский государственный университет» (ПГУ)

ПРОГРАММИРОВАНИЕ
НА ЯЗЫКЕ C++.
КУРСОВОЕ ПРОЕКТИРОВАНИЕ

Учебное пособие

Пенза
Издательство ПГУ
2023

УДК 004.67
П78

Р е ц е н з е н т ы :

начальник отдела разработки и тестирования средств
сетевой защиты ООО «Код безопасности» (г. Пенза)

Д. П. Дзюба;

кандидат технических наук, доцент кафедры вычислительной техники
Пензенского государственного университета

А. С. Бычков

А в т о р ы :

Л. В. Гурьянов, Л. С. Гурьянова, Е. А. Дзюба,
С. В. Самуйлов, А. Ю. Афонин

П78 Программирование на языке C++. Курсовое проектирование : учеб. пособие / Л. В. Гурьянов, Л. С. Гурьянова, Е. А. Дзюба [и др.]. – Пенза : Изд-во ПГУ, 2023. – 76 с.

ISBN 978-5-907666-67-2

Рассматриваются понятия и этапы разработки приложения на языке C++. Простыми примерами сопровождаются базовые принципы объектно ориентированного программирования, такие как инкапсуляция, наследование и полиморфизм. Задание на курсовое проектирование включает использование агрегации и композиции объектов различных классов, а также шаблонных классов библиотеки STL (vector, list, set, stack, queue). Моделирование программных средств осуществляется на языке UML с использованием диаграмм вариантов использования, классов и компонентов. Требования к документированию разработки соответствуют требованиям к курсовым проектам ПГУ.

Издание подготовлено на кафедре «Математическое обеспечение и применение ЭВМ» ПГУ и рекомендовано для обучающихся по направлению подготовки 09.03.04 «Программная инженерия» и 09.03.02 «Информационные системы и технологии».

УДК 004.67

ISBN 978-5-907666-67-2

© Пензенский государственный
университет, 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1. РАЗРАБОТКА ОБЪЕКТНО ОРИЕНТИРОВАННОГО ПРИЛОЖЕНИЯ НА ЯЗЫКЕ C++.....	6
1.1. Анализ предметной области и функциональных требований.....	6
1.1.1. Анализ предметной области	6
1.1.2. Анализ функциональных требований.....	7
1.1.3. Рекомендации к анализу функциональных требований	11
1.2. Проектирование	13
1.2.1. Наследование и полиморфизм.....	17
1.2.2. Наследование интерфейса.....	19
1.2.3. Композиция и агрегация	22
1.2.4. Проектирование программных средств курсового проекта	25
1.2.5. Рекомендации к проектированию структуры данных и программных средств	30
1.3. Реализация	31
1.3.1. Отображение графической информации в окне консольного приложения	33
1.3.2. Обработка исключительных ситуаций в курсовом проекте	40
1.3.3. Рисование фигур с помощью платформы .NET Framework	44
1.3.4. Рекомендации к реализации программных средств.....	53
1.4. Тестирование	54
2. СТРУКТУРА И ОФОРМЛЕНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ.....	57
2.1. Структура пояснительной записки	57
2.1.1. Титульный лист и техническое задание	57
2.1.2. Содержание и разделы разработки программных средств.....	60
2.2. Требования к оформлению пояснительной записки	62
2.2.1. Общие требования	62
2.2.2. Требования к оформлению перечислений	62
2.2.3. Требования к оформлению таблиц	63
2.2.4. Требования к оформлению рисунков	64
2.2.5. Требования к оформлению используемых источников.....	65
2.2.6. Требования к оформлению приложений	65

3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ ПО КУРСОВОМУ ПРОЕКТУ	66
3.1. Предметная область	66
3.2. Функциональные требования и требования к реализации.....	67
4. ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ И ЗАЩИТЫ КУРСОВОГО ПРОЕКТА.....	69
4.1. Раздел «Анализ предметной области и функциональных требований».....	69
4.2. Раздел «Проектирование»	69
4.3. Раздел «Реализация»	70
4.4. Раздел «Тестирование»	70
ПРИЛОЖЕНИЕ А	71
ПРИЛОЖЕНИЕ Б.....	73

ВВЕДЕНИЕ

Курсовое проектирование в форме выполнения курсовых проектов (КП) или курсовых работ (КР) является видом внеаудиторной академической работы обучающихся под руководством преподавателя, направленной на закрепление изучаемой дисциплины или учебного модуля в пределах часов, отводимых на изучение данной позиции учебного плана.

Целью курсового проектирования является овладение методикой или навыками самостоятельного решения конкретных задач на основе ранее приобретенных знаний, развитие у обучающихся как универсальных базовых, так и профессиональных компетенций в виде знаний, умений, навыков и способностей.

Курсовой проект – самостоятельная работа в письменной форме. Его обязательной составляющей является технический проект по заданной теме. В отличие от КР, Курсовой проект, помимо текстовой и расчетной части, обычно включает в себя графическую часть, которая состоит из чертежей, схем и таблиц.

Основными задачами курсового проектирования являются:

- углубление уровня освоения общекультурных и профессиональных компетенций;
- формирование умений и навыков самостоятельной организации учебно-исследовательской работы и применения полученных знаний для решения профессиональных задач;
- формирование умения работать с научной и справочной литературой, в том числе нормативными документами и стандартами;
- освоение современными методами поиска, обработки, использования и анализа информации;
- формирование необходимой культуры выполнения квалификационных работ.

1. РАЗРАБОТКА ОБЪЕКТНО ОРИЕНТИРОВАННОГО ПРИЛОЖЕНИЯ НА ЯЗЫКЕ C++.

1.1. Анализ предметной области и функциональных требований

1.1.1. Анализ предметной области

Предметная область разработки включает геометрические фигуры, указанные в варианте технического задания на курсовой проект. Пример предметной области приведен на рис. 1.

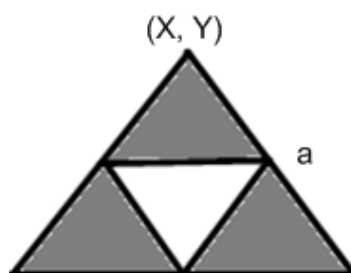


Рис. 1. Заданная геометрическая фигура

Модель предметной области – это диаграмма классов сущностей предметной области, в которой для каждой сущности (класса) определены наименование класса и описание данных, но отсутствует описание функций. Модель предметной области для геометрической фигуры (рис. 1) приведена на рис. 2.



Рис. 2. Модель предметной области

В данной модели сложная фигура представляет правильный треугольник, составленный из трех треугольников (закрашенные треугольники на рис. 1), и определяется координатами вершины (X, Y) и длиной стороны (a). Отдельный правильный треугольник определяется аналогично сложной фигуре.

1.1.2. Анализ функциональных требований

Анализ функциональных требований осуществляется на основе функций, заданных в техническом задании.

Описание данного раздела включает диаграмму вариантов использования (use case), сценариев вариантов использования и соответствующих им диаграмм последовательностей. Название вариантов использования должно совпадать с названиями функций в техническом задании, количество вариантов – не менее количества функций, заданных в техническом задании.

Пример 1. Предметная область – треугольники

В техническом задании на курсовой проект определены следующие функциональные требования:

- создать геометрическую фигуру (треугольник, сложная фигура);
- показать/скрыть фигуру;
- переместить фигуру.

Диаграмма вариантов использования, соответствующая этим требованиям, приведена на рис. 3. На диаграмме варианты «Показать треугольник» и «Скрыть треугольник» расширяют функциональность варианта «Переместить треугольник» (отношение *extends* – способ введения нового поведения в существующий вариант использования). Аналогичное отношение расширения используется для вариантов сложной фигуры. Это расширение обосновано алгоритмом функции «Переместить фигуру», который сначала удаляет геометрическую фигуру, а потом показывает ее на новом месте (относительно новой точки «привязки»).

Сценарии варианта использования «Переместить треугольник» и расширяющих его вариантов приведены в табл. 1–3.

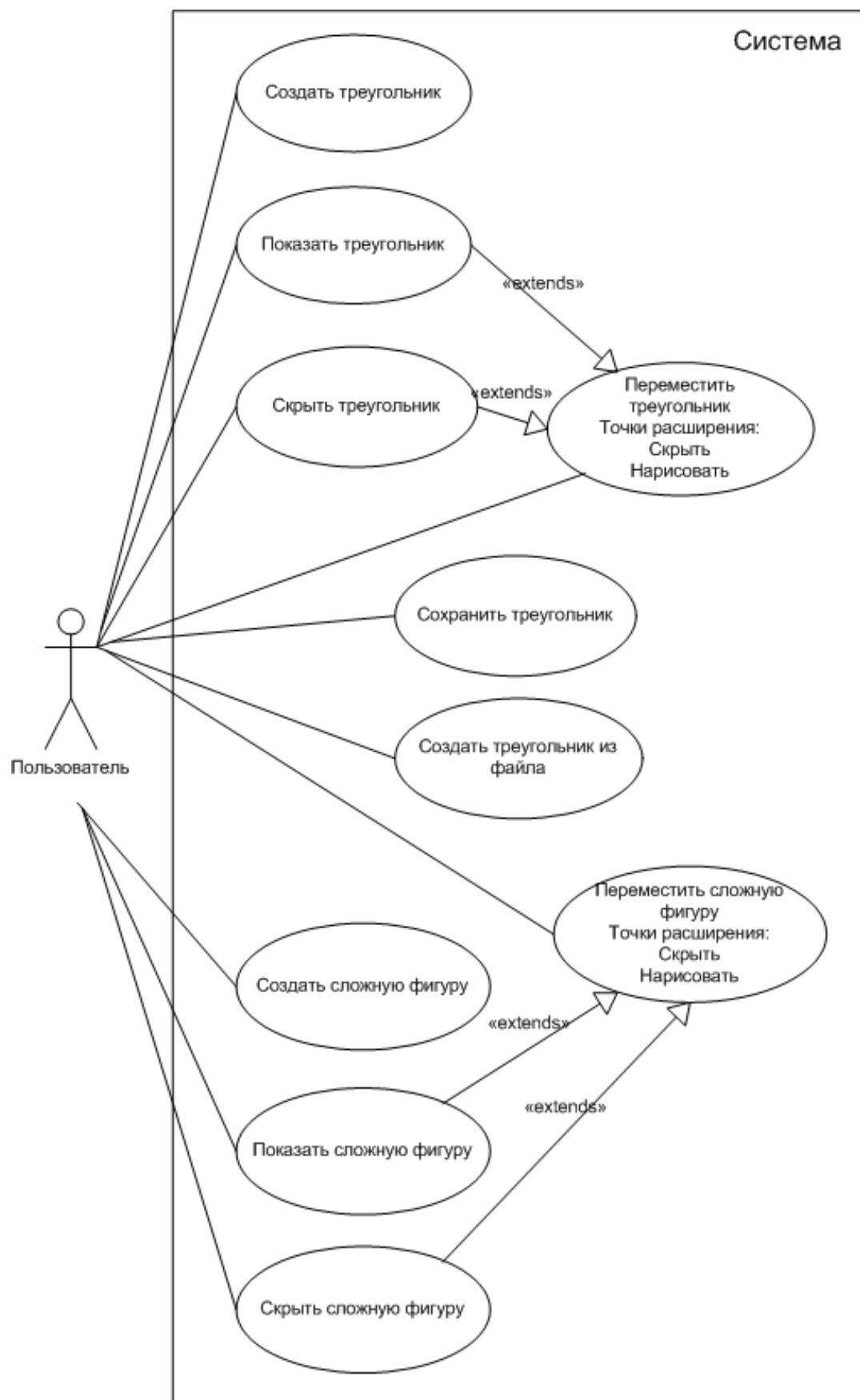


Рис. 3. Диаграмма вариантов использования для примера 1

Таблица 1

Сценарий варианта использования «Переместить треугольник»

Наименование: Переместить треугольник
ID: 4
Краткое описание: система удаляет изображение треугольника и рисует его в новых координатах
Действующие лица: пользователь
Предусловие: треугольник создан и нарисован
Основной поток: 1. Пользователь задает новые координаты X, Y треугольника. Точка расширения «Скрыть». 2. Система переопределяет координаты треугольника. Точка расширения «Нарисовать»
Постусловие: треугольник перемещен

Таблица 2

Сценарий варианта использования «Скрыть треугольник»

Наименование: Скрыть треугольник
ID: 3
Краткое описание: система удаляет изображение треугольника
Действующие лица: пользователь
Предусловие: треугольник создан и нарисован
Основной поток: 1. Пользователь инициирует удаление изображения треугольника. 2. Система рисует изображение треугольника цветом фона
Постусловие: изображение треугольника скрыто

Таблица 3

Сценарий варианта использования «Показать треугольник»

Наименование: Показать треугольник
ID: 2
Краткое описание: система рисует изображение треугольника с координатами вершины (X, Y) и заданной стороной (a)
Действующие лица: пользователь
Предусловие: треугольник создан
Основной поток: 1. Пользователь инициирует рисование треугольника с координатами вершины (X, Y) и стороной (a) 2. Система рисует изображение треугольника с заданными координатами вершины (X, Y) и заданной стороной (a)
Постусловие: треугольник нарисован

Пример 2. Предметная область – треугольники, список

В техническом задании на курсовой проект определены следующие функциональные требования:

- создать геометрическую фигуру (треугольник, сложная фигура);
- добавить фигуру в список;
- показать фигуры из списка;
- удалить список фигур.

Диаграмма вариантов использования, соответствующая этим требованиям, приведена на рис. 4. На диаграмме вариант «Создать треугольник» включает вариант «Добавить фигуру в список» (отношение *include/uses* выносит шаги, общие для нескольких вариантов в отдельный вариант, который потом включается в остальные варианты использования).

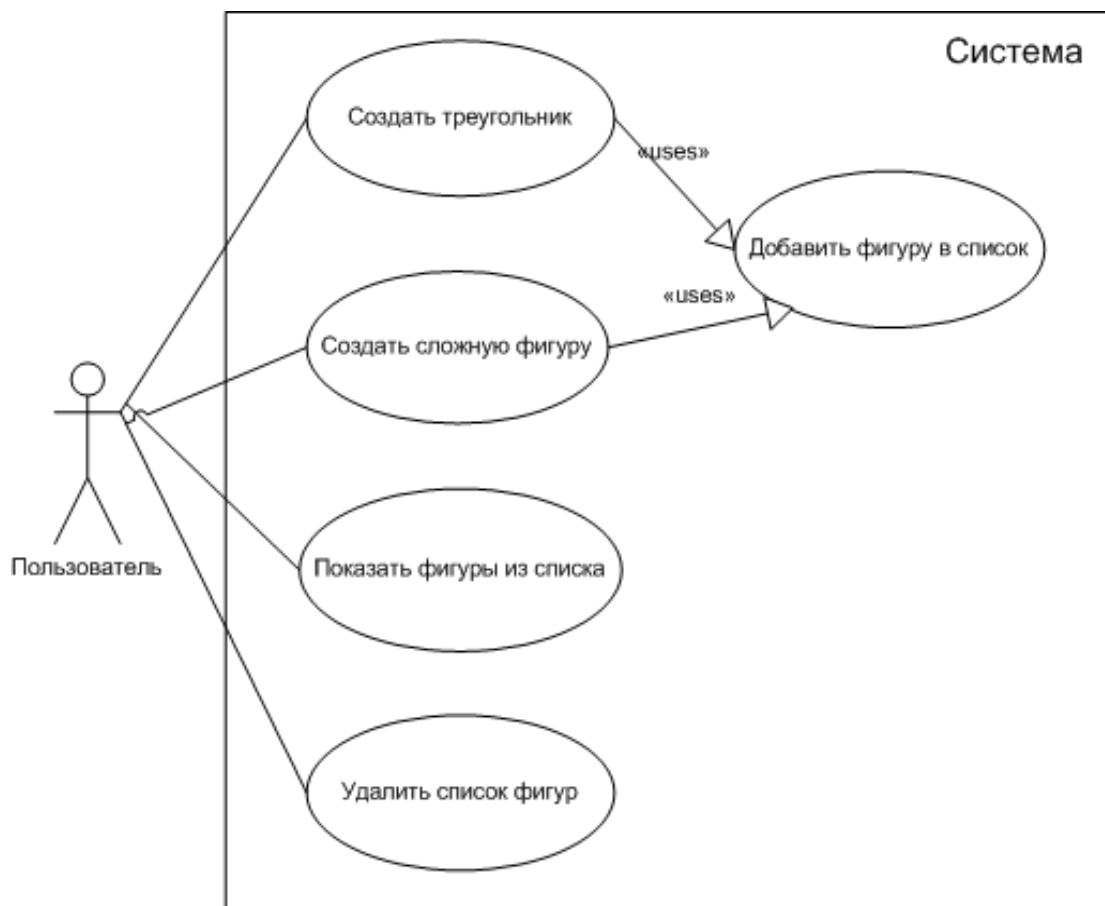


Рис. 4. Диаграмма вариантов использования для примера 2

Сценарии варианта «Создать треугольник» и включаемого варианта использования «Добавить фигуру в список» приведены в табл. 4, 5.

Таблица 4

Сценарий варианта использования «Создать треугольник»

Наименование: Создать треугольник
ID: 1
Краткое описание: система создает объект «Треугольник» с координатами вершины (X, Y) и заданной стороной (a) и помещает его в список
Действующие лица: пользователь
Предусловие: треугольник создан
Основной поток: 1. Пользователь задает атрибуты треугольника (координаты вершины X, Y и сторону (a)) 2. Система создает объект «Треугольник» с заданными атрибутами 3. Include (Добавить фигуру в список)
Постусловие: треугольник создан

Таблица 5

Сценарий варианта использования «Добавить фигуру в список»

Наименование: Добавить фигуру в список
ID: 3
Краткое описание: система добавляет созданный объект «Треугольник» или объект «Сложная фигура» в конец списка фигур
Действующие лица: система
Предусловие: создан объект «Треугольник» или объект «Сложная фигура»
Основной поток: 1. Система проверяет, создан ли список фигур: 1.1. Если список существует, то система добавляет созданный объект геометрической фигуры в конец списка фигур. 1.2. Иначе: 1.2.1. Система создает список из одного элемента – созданного объекта геометрической фигуры
Постусловие: фигура добавлена в список

1.1.3. Рекомендации к анализу функциональных требований

Для выполнения анализа функциональных требований следует:

- внимательно прочитать требования технического задания. Если возникают вопросы, обсудить их с преподавателем, и только после этого приступить к разработке программных средств;
- дать краткое пояснение к модели предметной области – перечислить наименования сущностей и их атрибутов;

- помнить, что основной поток сценария варианта использования описывает, **что** делает система, а **не как** она это делает;
- выполнить диаграммы вариантов использования таким образом, чтобы они были простыми и понятными (без необходимости не применять отношения расширения или включения);
- при разработке сценариев вариантов использования, связанных отношениями расширения или включения, в пояснительной записке приводить сценарии как для основного варианта, так и для расширяющего или включаемого вариантов;
- для графического отображения сценариев использовать диаграммы последовательностей (пример, соответствующий варианту использования «Создать треугольник», приведен на рис. 5).

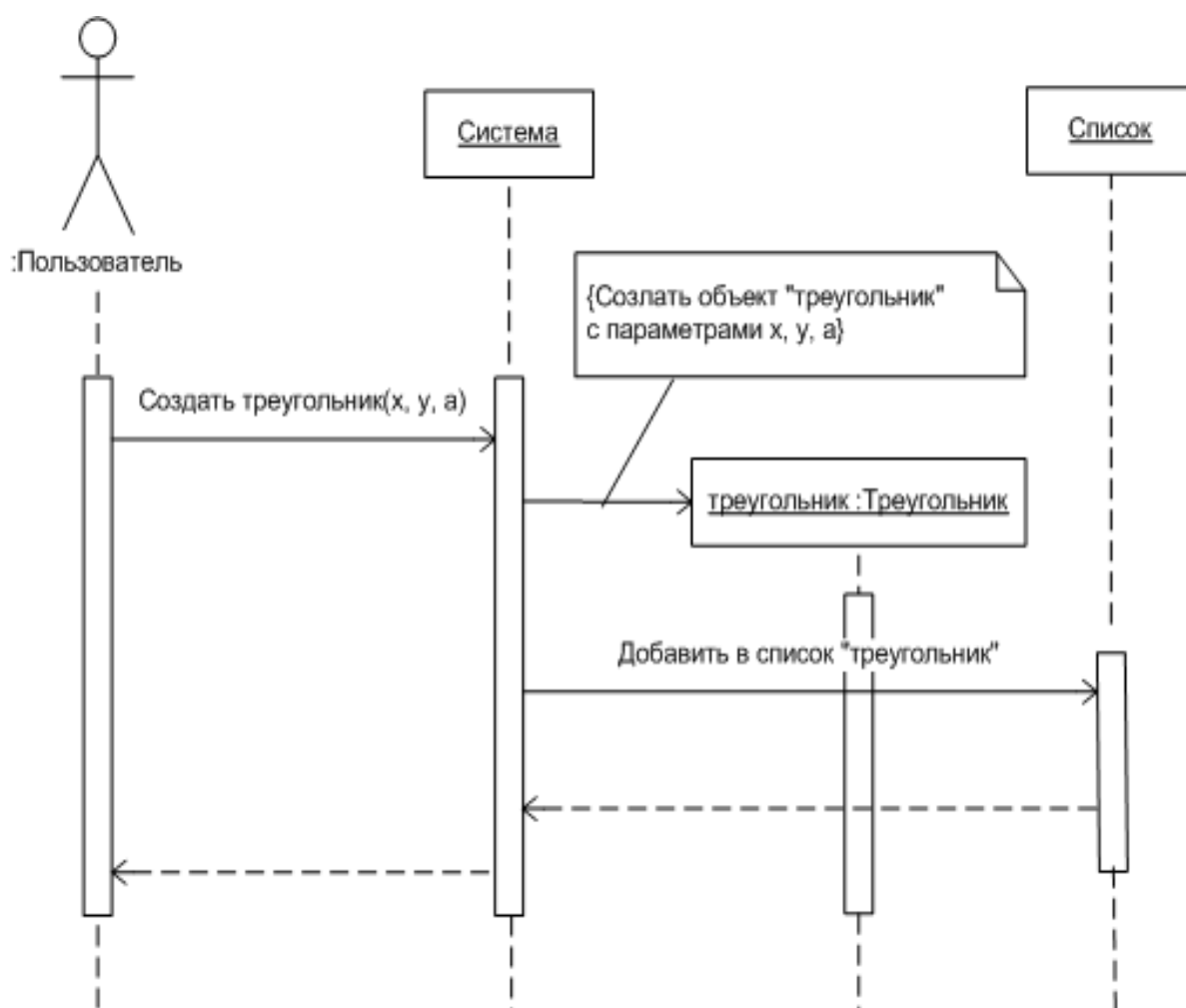


Рис. 5. Диаграмма последовательности для варианта использования «Создать треугольник»

1.2. Проектирование

В данном разделе обычно рассматривают требования и решения, связанные с проектированием интерфейса пользователя, структуры данных и программных средств (ПС) разрабатываемого программного обеспечения.

Объектно ориентированное проектирование включает разработку классов и их взаимодействия. В структуре программных средств широко используются принципы объектного подхода, такие как наследование и полиморфизм.

Моделирование структуры ПС осуществляется с помощью диаграммы классов. Особенности объектно ориентированной модели:

- модель содержит изображение программных классов с реальными названиями классов, атрибутов и функций (как в программном коде);
- в обозначении используются символы, соответствующие спецификаторам доступа к членам и функциям класса (+ – **public**; - – **private**; # – **protected**), а также отношениям между классами (определенный тип линии со своим окончанием);
- в модели используются классы предметной области.

Пример 3. UML, класс Figure

Рассмотрим пример моделирования класса **Figure** в среде **Microsoft Visio**.

1. Создаем пустую модель программного проекта. Для этого в меню **Файл** последовательно выбираем: **Создать** → **Программное обеспечение и базы данных** → **Схема модели UML** (рис. 6).

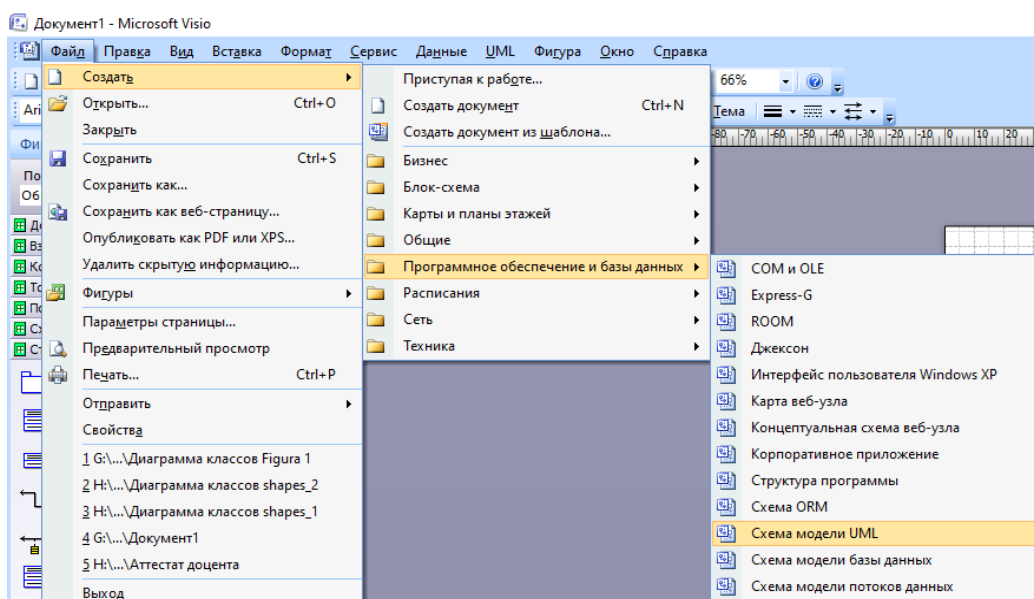


Рис. 6. Создание модели программного проекта

2. Среди моделей программного обеспечения выбираем **Статическая структура UML** – диаграмму классов и перетаскиваем в окно формирования модели изображение **Класс**.

3. В контекстном меню фигуры **Класс** выбираем **Свойства** (рис. 7).

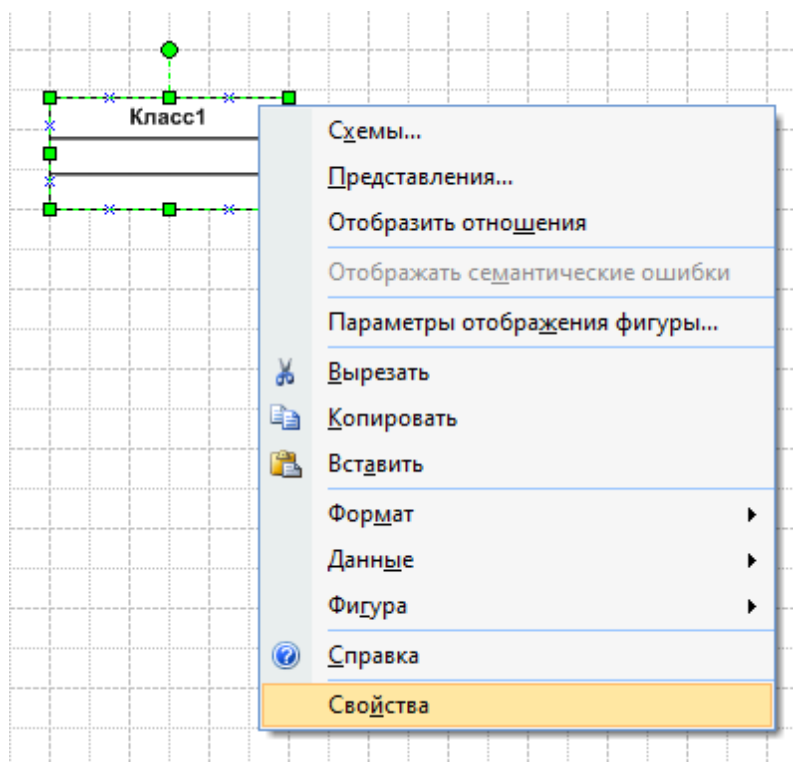


Рис. 7. Выбор в контекстном меню

4. В окне **Свойства класса UML** последовательно задаем:

- в категории **Класс: Имя** класса (**Figure**) и его свойства (рис. 8);
- в категории **Атрибуты**: атрибуты класса (**x**, **y**) с указанием их наименований, типов и видимости (рис. 9);

- в категории **Операции**: функции класса с указанием их наименования, типа возвращаемого значения, видимости и полиморфизма – виртуальная или чисто виртуальная функция (рис. 10). Задание параметров функции показано на рис. 11.

Для задания параметров функции выделяем ее (на рис. 10 – конструктор **Figure**) и нажимаем на кнопку **Свойства**. Далее в открывшемся окне **Свойства операции UML** (рис. 11) выбираем категорию **Параметры** и задаем необходимые параметры с указанием их наименования (в данном случае **x**, **y**), типа и назначения: входные (**in**) или выходные (**out**).

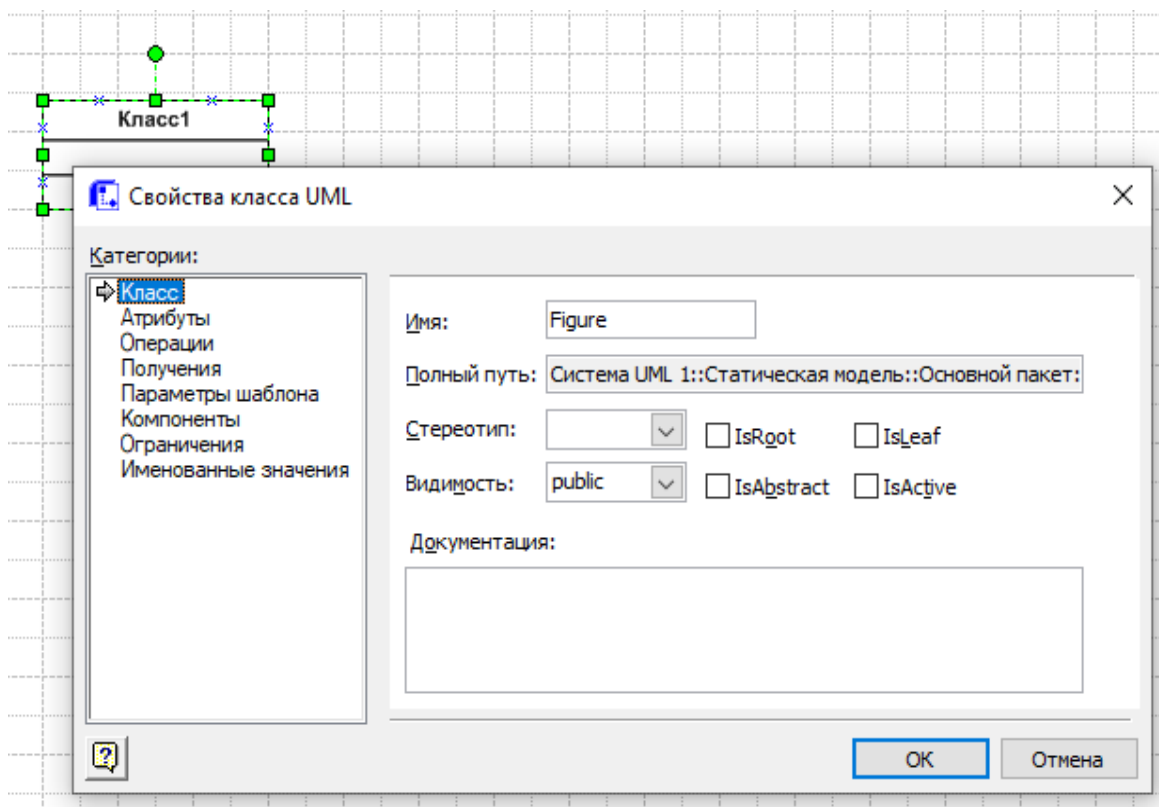


Рис. 8. Определение свойств класса (категория **Класс**)

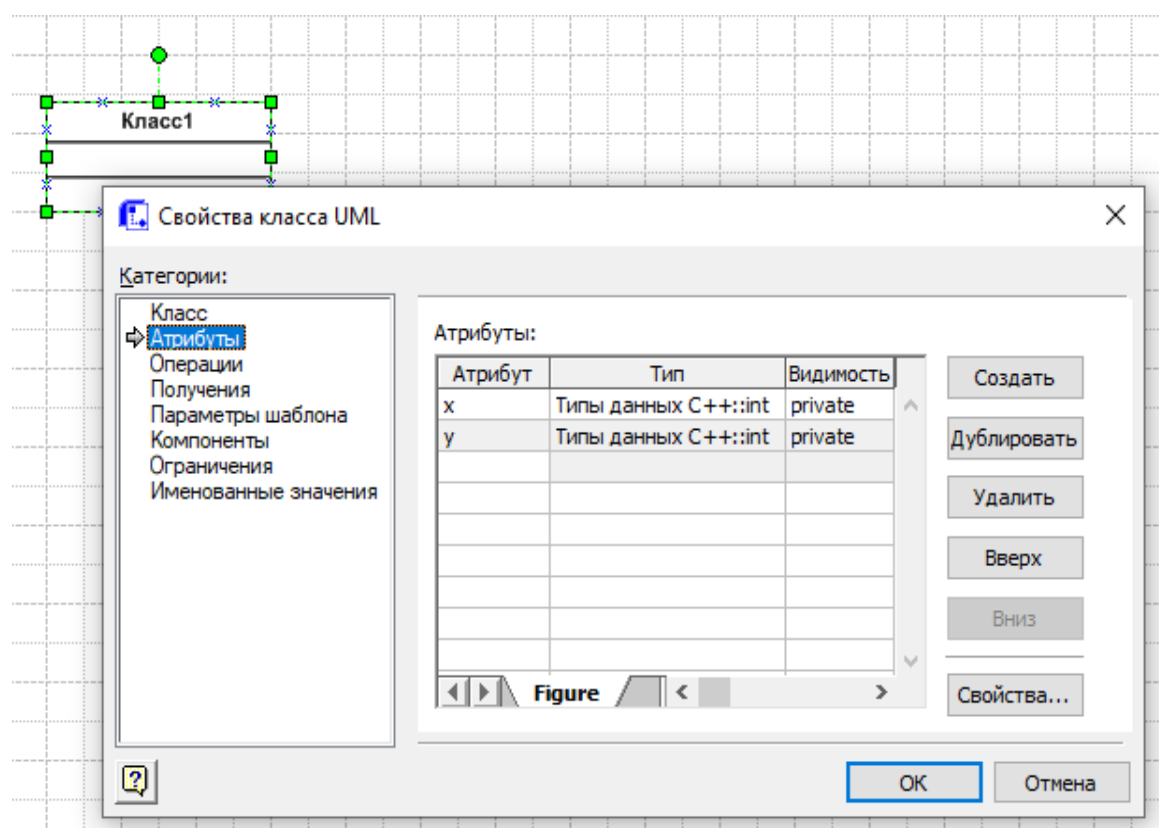


Рис. 9. Определение атрибутов класса (категория **Атрибуты**)

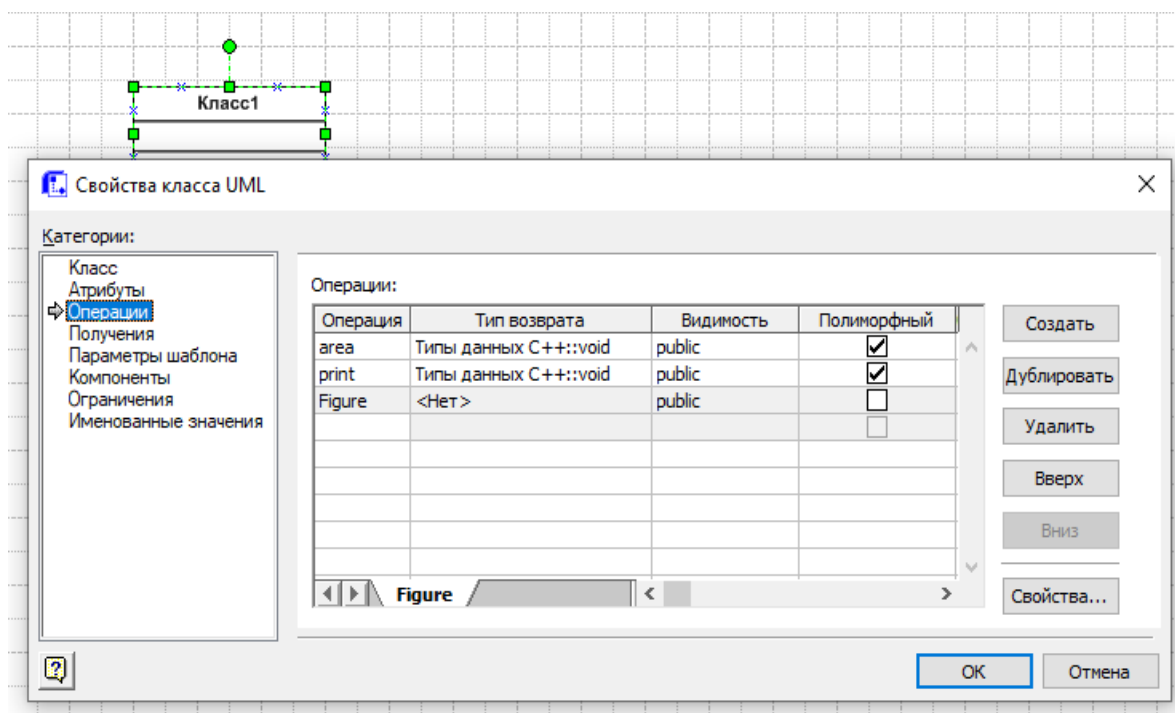


Рис. 10. Определение функций класса (категория **Операции**)

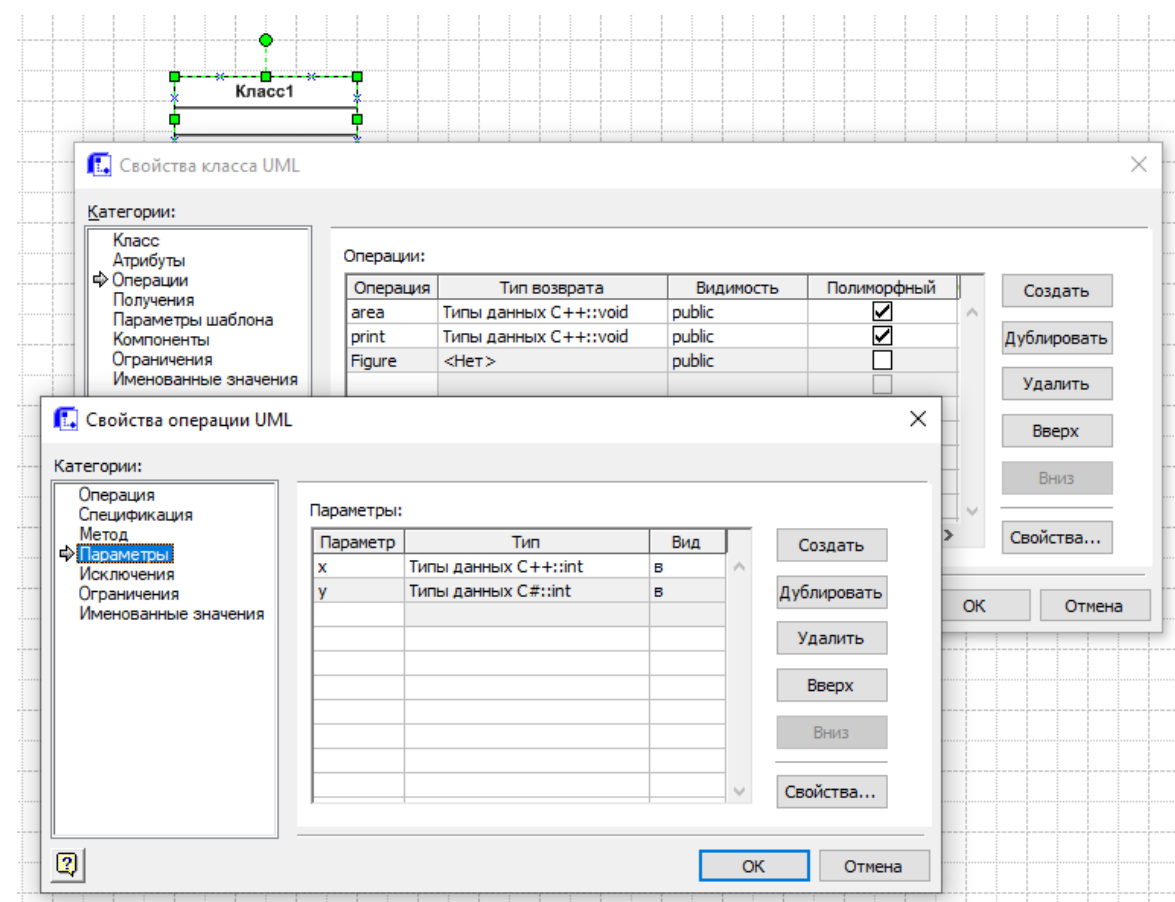


Рис. 11. Определение функций класса (категории **Операции**, **Параметры**)

Сформированное изображение класса **Figure** и модель приложения показаны на рис. 12.

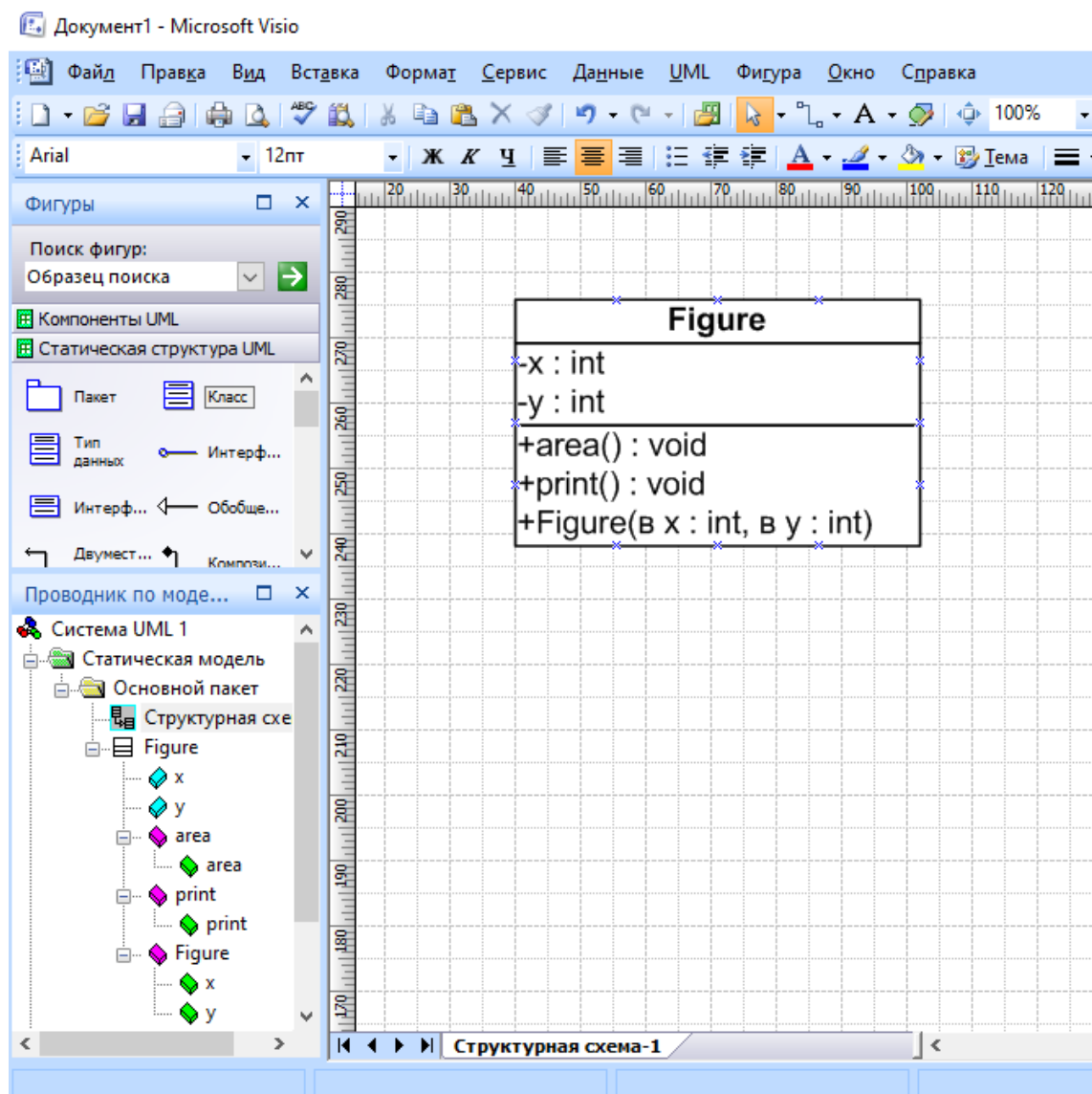


Рис. 12. Сформированный класс **Figure** и статическая модель программного проекта

1.2.1. Наследование и полиморфизм

Диаграмма классов для предметной области, содержащей прямоугольник и окружность, приведена на рис. 13.

Классы Окружность (**Circle**) и Прямоугольник (**Rectangle**) наследуют из базового класса **Figure** координаты центра фигуры (**x**, **y**) и реализуют виртуальную функцию **area** (вычисления площади фигуры) базового класса своим собственным алгоритмом.

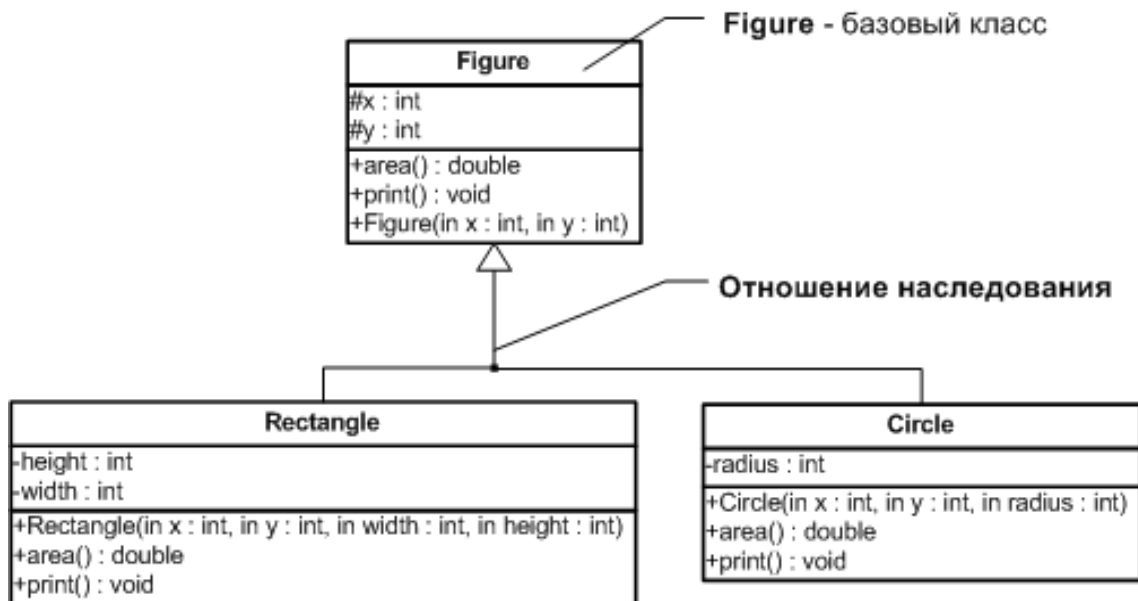


Рис. 13. Диаграмма классов для предметной области
Прямоугольник и Окружность

Далее приведем пример реализации данной структуры на языке C++.

Пример 4. Наследование и полиморфизм

```

#include <iostream>

using namespace std;

class Figure {
public:
    Figure(int _x, int _y) : x(_x), y(_y) {}
    virtual double area() const { return 0; };
    virtual void print() const {
        cout << x << ", " << y << ", ";
    }
protected:
    int x, y;
};

class Rectangle: public Figure {
public:
    Rectangle(int x, int y, int a, int b) : Figure(x, y),
                                            height(a), width(b) {}
    double area() const override { return height*width; }
    void print() const override {
        cout << "Rectangle: ";
        Figure::print();
        cout << height << ", " << width;
    }
};
  
```

```

    }
private:
    int height, width;
};

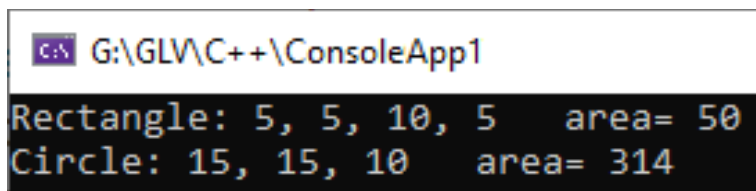
class Circle: public Figure {
public:
    Circle(int x, int y, int a) : Figure(x, y), radius(a) {}
    double area() const override { return 3.14*radius*radius;
}

    void print() const override {
        cout << "Circle: ";
        Figure::print();
        cout<< radius;
    }
private:
    int radius;
};

int main(){
    Figure* s[2];
    s[0] = new Rectangle(5, 5, 10, 5);
    s[1] = new Circle(15, 15, 10);
    s[0]->print();
    cout <<"    area= "<<s[0]->area() << endl;
    s[1]->print();
    cout <<"    area= "<<s[1]->area() << endl;
    system("pause");
    return 0;
}

```

Результат работы программы приведен на рис. 14.



```

G:\GLV\C++\ConsoleApp1
Rectangle: 5, 5, 10, 5    area= 50
Circle: 15, 15, 10    area= 314

```

Рис. 14. Результат работы программы (Пример 4)

1.2.2. Наследование интерфейса

Диаграмма классов для предметной области, содержащей базовый абстрактный класс, прямоугольник и окружность, приведена на рис. 15.

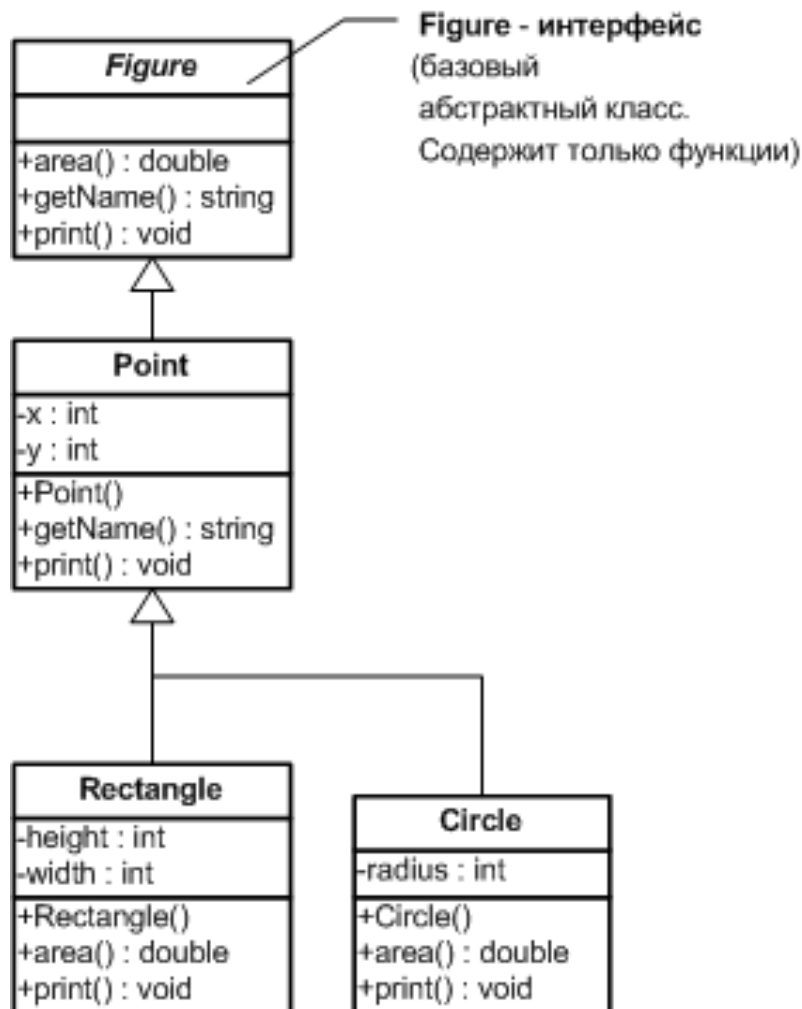


Рис. 15. Диаграмма классов, построенная с наследованием интерфейса, для предметной области Прямоугольник и Окружность

Базовый абстрактный класс **Figure** не содержит членов, а включает только чисто виртуальные функции **getName**, **area**, **print**. Класс **Point** наследуется от класса **Figure**. Классы Окружность (**Circle**) и Прямоугольник (**Rectangle**) наследуют из класса **Point** координаты центра фигуры (**x**, **y**) и реализуют виртуальные функции **getName**, **area**, **print** своими собственными алгоритмами.

Рассмотрим пример реализации данной структуры.

Пример 5. Интерфейс

```

#include <iostream>
#include <string>
using namespace std;
class Figure {
public:
    virtual string getName() const = 0;
  
```

```

        virtual double area()    const = 0;
        virtual void  print()    const = 0;
};

class Point : public Figure {
public:
    Point(int a = 0, int b = 0) : x(a), y(b) {}
    string getName() const override { return "Point: "; }
    void  print() const override {
        cout << x << ", " << y << ", ";
    }
protected:
    int x, y;
};

class Circle : public Point {
public:

    Circle(int x, int y, int a) : Point(x, y), radius(a) {}
    string getName() const override { return "Circle: "; }
    double area() const override { return 3.14 * radius * radius;
}
    void print() const override {
        Point::print();
        cout << radius<<endl;
    }
private:
    int radius;
};

class Rectangle : public Point {
public:
    Rectangle(int x,int y,int a,int b): Point(x, y), height(a),
                                         width(b)
                                         {}
    string getName() const override { return "Rectangle: "; }
    double area() const {return static_cast<dou-
ble>(height)*width;}
    void  print() const override {
        Point::print();
        cout << height << ", " << width<<endl;
    }
private:
    int height, width;
};

int main()
{

```

```

    Point* p[3];
    p[0] = new Rectangle(20, 30, 10, 50);
    p[1] = new Circle(100, 100, 10);
    cout << p[0]->getName(); p[0]->print();
    cout<< "area: "<< p[0]->area() << endl;
    cout << p[1]->getName(); p[1]->print();
    cout<< "area: "<< p[1]->area() << endl;
    system("pause");
    return 0;
}

```

Результат работы программы приведен на рис. 16.

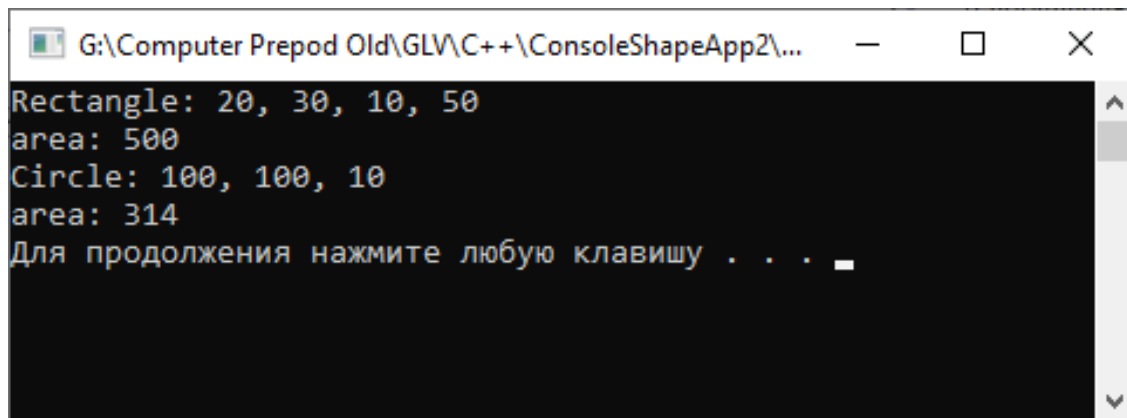


Рис. 16. Результат работы программы (Пример 5)

1.2.3. Композиция и агрегация

Семантику *агрегации* можно определить следующим образом:

- агрегат может существовать как независимо от частей, так и вместе с ними;
- части могут существовать независимо от агрегата;
- агрегат является в некотором смысле неполным в случае отсутствия некоторых частей;
- части могут принадлежать одновременно нескольким агрегатам.

Семантику *композиции* можно определить таким образом:

- одновременно части могут принадлежать только одному композиту, совместное владение частями невозможно;
- композит обладает исключительной ответственностью за все свои части – это означает, что он отвечает за их создание и уничтожение;

– КОМПОЗИТ может высвобождать части, передавая ответственность за них другому объекту;

– в случае уничтожения композита он должен или уничтожить все свои части, или передать ответственность за них другому объекту.

Реализация агрегации и композиции на языке C++ рассмотрена на примере классов сложной фигуры, состоящей из окружности и прямоугольника: класс **AgrigateFigures** – агрегация фигур, класс **CompositionFigures** – композиция.

Пример 6. Композиция и агрегация

```
#include <iostream>
using namespace std;

class Figure {
public:
    Figure(int _x, int _y) :x(_x), y(_y) {}
    virtual double area() const { return 0; }
    virtual void draw () const {
        cout << x << ", " << y << ", ";
    }
protected:
    int x, y;
};

class Rectangle : public Figure {
public:
    Rectangle(int x, int y, int a, int b) : Figure(x, y),
                                             height(a), width(b){}
    double area() const override { return height * width; }
    void draw() const override {
        cout << "Rectangle: "; Figure::draw();
        cout << height << ", " << width << endl;
    }
private:
    int height, width;
};

class Circle : public Figure {
public:
    Circle(int x, int y, int a) : Figure(x, y), radius(a) {}
    double area() const { return 3.14 * radius * radius; }
    void draw() const override {
        cout << "Circle: "; Figure::draw();
        cout << radius << endl;
    }
}
```

```

private:
    int radius;
};

//Агрегация
class AgrFigure {
public:
    AgrFigure(Circle* _c, Rectangle* _r) : c(_c), r(_r) {}
    void draw() const {
        cout << "Agrigate : " << endl;
        c->draw();
        r->draw();
    }
private:
    Circle* c;
    Rectangle* r;
};

//Композиция
class CompFigure {
public:
    CompFigure(int x, int y, int radius, int height, int width) {
        c = new Circle(x, y, radius);
        r = new Rectangle(x, y, height, width);
    }
    void draw() const {
        cout << "Composition : " << endl;
        c->draw();
        r->draw();
    }
private:
    Circle* c;
    Rectangle* r;
};

int main()
{
    Figure* s[2];
    s[0] = new Rectangle(10, 10, 10, 5);
    s[1] = new Circle(100, 100, 10);
    s[0]->draw();
    cout << s[0]->area() << endl;
    s[1]->draw();
    cout << s[1]->area() << endl;
    Circle* c1 = new Circle(50, 50, 50);
    Rectangle* r1 = new Rectangle(100, 100, 40, 60);

```

```

    AgrFigure* argF = new AgrFigure(c1, r1);
    argF->draw();
    cout << endl;

    CompFigure compF(30, 30, 30, 50, 100);
    compF.draw();

}

```

Результат работы программы приведен на рис. 17.

```

Консоль отладки Microsoft Visual Studio
Rectangle: 10, 10, 10, 5
50
Circle: 100, 100, 10
314
Agrigate :
Circle: 50, 50, 50
Rectangle: 100, 100, 40, 60

Composition :
Circle: 30, 30, 30
Rectangle: 30, 30, 50, 100

G:\Computer Prepod Old\GLV\C++\ConsoleCompAgrApp1\Debug\ConsoleCompAgrApp1.exe
(процесс 8360) завершил работу с кодом 0. Чтобы автоматически закрывать консоль
при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...

```

Рис. 17. Результат работы программы (Пример 6)

1.2.4. Проектирование программных средств курсового проекта

В основе моделирования структуры ПС курсового проекта лежит модель предметной области (классы фигур) и модель хранения информации о создаваемых фигурах (класс хранения).

Структура программных средств для предметной области «Прямоугольник, вписанный в окружность» приведена на рис. 18.

На приведенной диаграмме классы Окружность (**Circle**) и Прямоугольник (**Rectangle**) наследуют из базового класса **Figure** координаты центра фигуры (**x**, **y**). Класс сложной фигуры (**ComplexFigure**) содержит композицию из объектов классов **Circle** и **Rectangle**.

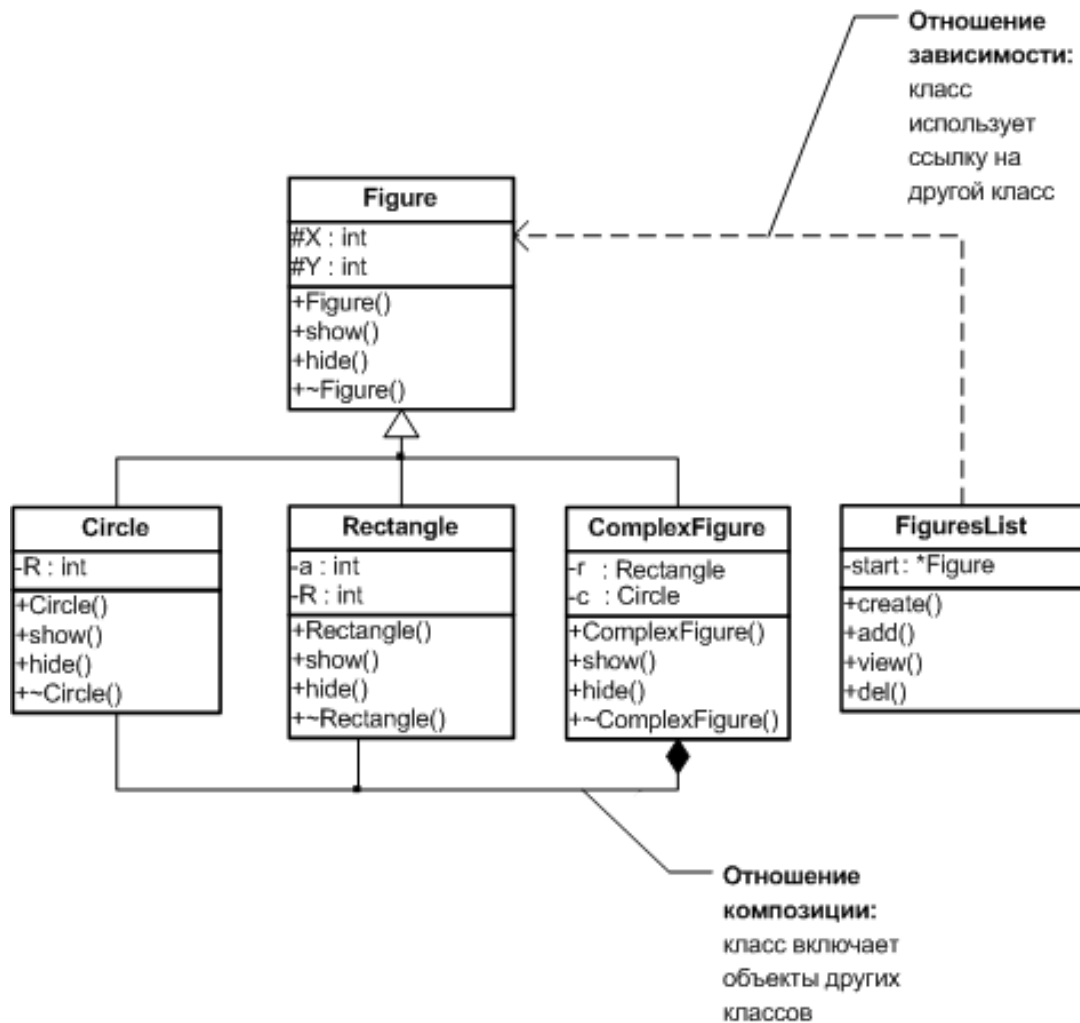


Рис. 18. Диаграмма классов для предметной области
«Прямоугольник, вписанный в окружность»

Класс хранения (на диаграмме классов – список) содержит описание структуры данных, которая использует тип базового класса фигур (на рис. 18 – `start: *Figure`).

Для реализации класса хранения информации о фигурах используются контейнерные классы C++: `vector`, `list`, `set`, `stack`, `queue`.

Далее приведен пример использования контейнерных классов.

Пример 7. Контейнеры

```

#include <iostream>
#include <vector>
#include <list>
#include <stack>
#include <set>
#include <queue>
using namespace std;
  
```

```

class Figure {
public:
    Figure(int _x, int _y) : x(_x), y(_y) {}
    virtual double area() const { return 0; };
    virtual void print() const {
        cout << x << ", " << y << ", ";
    }
protected:
    int x, y;
};

class Rectangle: public Figure {
public:
    Rectangle(int x, int y, int a, int b) : Figure(x, y),
                                            height(a), width(b) {}
    double area() const override { return height*width; }
    void print() const override {
        cout << "Rectangle: ";
        Figure::print();
        cout<<height << ", " << width;
    }
private:
    int height, width;
};

class Circle: public Figure {
public:
    Circle(int x, int y, int a) : Figure(x, y), radius(a) {}
    double area() const override { return 3.14*radius*radius;
}

    void print() const override {
        cout << "Circle: ";
        Figure::print();
        cout<< radius;
    }
private:
    int radius;
};

int main()
{
    Figure* s[2];
    s[0] = new Rectangle(5, 5, 10, 5);
    s[1] = new Circle(15, 15, 10);
    s[0]->print(); cout << "    area= "<<s[0]->area() << endl;
    s[1]->print(); cout << "    area= "<<s[1]->area() << endl;
}

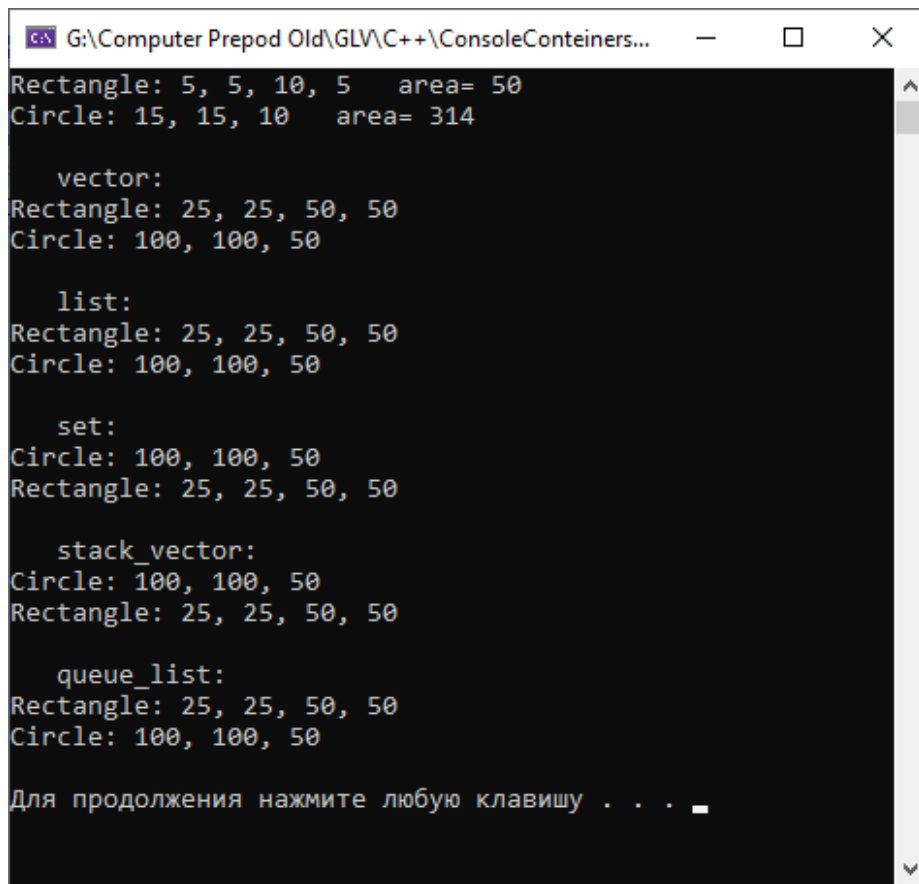
```

```

cout << endl;
cout << "    vector: " << endl;
vector<Figure*> v;
v.push_back(new Rectangle(25, 25, 50, 50));
v.push_back(new Circle(100, 100, 50));
for (auto i : v) {
    i->print(); cout << endl;
}
cout << endl;
cout << "    list: " << endl;
list<Figure*> ls;
ls.push_back(new Rectangle(25, 25, 50, 50));
ls.push_back(new Circle(100, 100, 50));
for (auto i : ls) {
    i->print(); cout << endl;
}
cout << endl;
cout << "    set: " << endl;
set<Figure*> st;
st.insert(new Rectangle(25, 25, 50, 50));
st.insert(new Circle(100, 100, 50));
for (auto i : st) {
    i->print(); cout << endl;
}
cout << endl;
cout << "    stack_vector: " << endl;
stack<Figure*, vector<Figure*> > sv;
sv.push(new Rectangle(25, 25, 50, 50));
sv.push(new Circle(100, 100, 50));
while(!sv.empty()) {
    sv.top()->print(); cout << endl; sv.pop();
}
cout << endl;
cout << "    queue_list: " << endl;
queue<Figure*, list<Figure*> > q;
q.push(new Rectangle(25, 25, 50, 50));
q.push(new Circle(100, 100, 50));
while (!q.empty()) {
    q.front()->print(); cout << endl; q.pop();
}
system("pause");
return 0;
}

```

Результат работы программы приведен на рис. 19.



```
G:\Computer Prepod Old\GLV\C++\ConsoleContainers...
Rectangle: 5, 5, 10, 5   area= 50
Circle: 15, 15, 10   area= 314

vector:
Rectangle: 25, 25, 50, 50
Circle: 100, 100, 50

list:
Rectangle: 25, 25, 50, 50
Circle: 100, 100, 50

set:
Circle: 100, 100, 50
Rectangle: 25, 25, 50, 50

stack_vector:
Circle: 100, 100, 50
Rectangle: 25, 25, 50, 50

queue_list:
Rectangle: 25, 25, 50, 50
Circle: 100, 100, 50

Для продолжения нажмите любую клавишу . . .
```

Рис. 19. Результат работы программы (Пример 7)

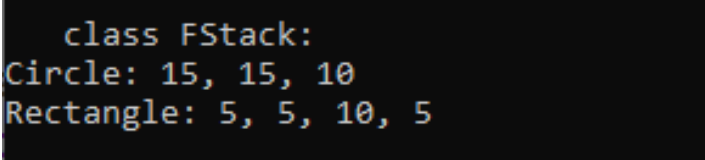
Добавим описание класса хранения (на примере стека) в программу примера 7:

```
class FStack {
private:
    stack<Figure*, list<Figure*> > sf;
public:
    FStack(){}
    void add(Figure* f) { sf.push(f); }
    void print() {
        while (!sf.empty()) {
            sf.top()->print(); cout << endl; sf.pop();
        }
        cout << endl;
    }
    ~FStack() {
        while (!sf.empty()) sf.pop();
    }
};
```

В главную программу добавим тесты для созданного класса **FStack**:

```
cout << "   class FStack: " << endl;
FStack fstack;
fstack.add(s[0]);
fstack.add(s[1]);
fstack.print();
```

Результат использования класса **FStack** приведен на рис. 20.



```
class FStack:
Circle: 15, 15, 10
Rectangle: 5, 5, 10, 5
```

Рис. 20. Результат работы программы (Пример 7. Класс **FStack**)

Спецификация классов

Спецификация класса – это краткое описание класса, которое включает:

- название класса;
- назначение класса (если класс базовый, абстрактный или наследуемый, то это надо указать);
- описание членов класса с указанием наименования и типа;
- описание функций класса с указанием наименования и параметров.

Например, спецификацию класса **Окружность** можно описать так:

Название класса: **Circle**.

Назначение: класс фигуры **Окружность**, наследник класса **Figure**.

Члены класса: **radius: int** – радиус окружности.

Функции класса:

- **Circle (int, int, int)** – конструктор с координатами центра и радиуса окружности;
- **~ Circle** – деструктор;
- **show()** – показать окружность;
- **hide()** – скрыть окружность.

1.2.5. Рекомендации к проектированию структуры данных и программных средств

Для проектирования программных средств следует:

- хорошо изучить модель предметной области и функциональные требования;

- на основании модели предметной области и диаграммы вариантов использования сформировать классы программных средств;
- определить отношения между классами;
- в описание раздела включить обоснование выбранной структуры ПС, диаграмму классов, спецификацию всех классов созданного программного приложения;
- диаграмму классов, кроме рисунка в разделе «Проектирование», необходимо выполнить в формате А3 и поместить в графическую часть пояснительной записки (Приложение Б).

1.3. Реализация

Модель реализации – это часть проектной модели, занимающаяся вопросами реализации. Она определяет, как проектные элементы представляются артефактами и как эти артефакты развертываются на узлах вычислительной системы. Артефакты представляют описания реальных программных сущностей, а узлы представляют описания оборудования или сред выполнения, в которых эти сущности развертываются. Некоторые примеры артефактов:

- исходные файлы;
- исполняемые файлы;
- сценарии;
- таблицы баз данных;
- документы;
- результаты процесса разработки, например UML-модель.

UML предлагает стандартные стереотипы артефактов, представляющих разные типы файлов (табл. 6).

Таблица 6

Стереотипы артефактов

Стереотип артефакта	Семантика
file	Физический файл
executable	Исполняемый программный файл
source	Исходный файл, который может быть скомпилирован в исполняемый файл
document	Универсальный файл, содержащий некоторую информацию
library	Статическая или динамическая библиотека (DLL)

Модель реализации состоит из диаграмм компонентов, показывающих, как артефакты представляют компоненты, и диаграммы развертывания, показывающей физические вычислительные узлы (компьютеры), на которых будут развертываться программные компоненты.

В курсовом проекте текст раздела Реализация должен содержать ссылки на Приложение А и диаграмму компонентов. Например так: «Код программы для предметной области «Окружность и прямоугольник» с графическим выводом фигур в окне консольного приложения приведен в Приложение А, диаграмма компонентов – на рис. __».

Пример 8. Диаграмма компонентов

Диаграмма компонентов для предметной области «Окружность и прямоугольник» с графическим выводом фигур в окне консольного приложения приведена на рис. 21.

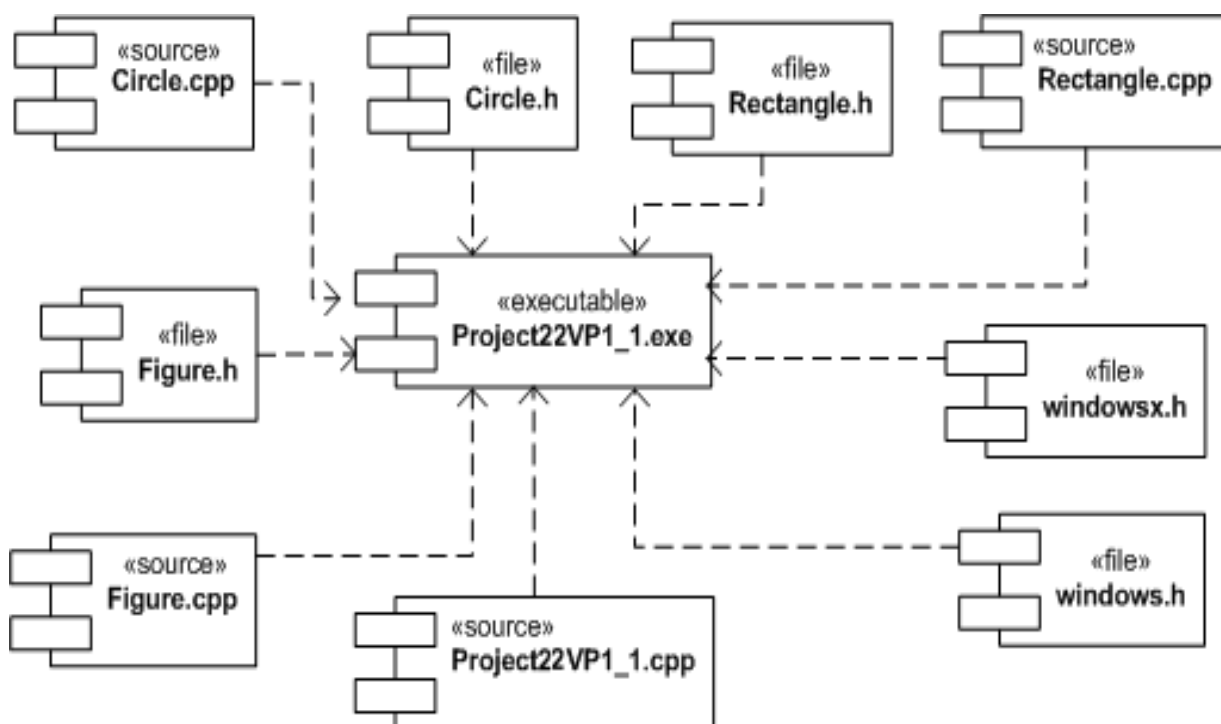


Рис. 21. Диаграмма компонентов для предметной области «Окружность и прямоугольник»

Описание компонентов приведено в табл. 7.

Компоненты для предметной области «Окружность и прямоугольник»

Компонент	Назначение
Figure.h	Заголовочный файл класса Figure
Figure.cpp	Исходный файл класса Figure
Circle.h	Заголовочный файл класса Circle
Circle.cpp	Исходный файл класса Circle
Rectangle.h	Заголовочный файл класса Rectangle
Rectangle.cpp	Исходный файл класса Rectangle
windows.h	Системный заголовочный файл
Windowsx.h	Системный заголовочный файл
Project22vp1_1.cpp	Главная программа
Project22vp1_1.exe	Исполняемый файл приложения

1.3.1. Отображение графической информации в окне консольного приложения

1.3.1.1. Основы графики

Система графических координат: $(0, 0)$ – левый верхний угол окна отображения, координата X растет вправо, Y – вниз. Для отображения графической информации в окне консольного приложения необходимо получить доступ и создать необходимые ресурсы:

- идентификатор окна: тип **HWND**;
- координаты прямоугольного окна: тип **RECT**, функция **GetClientRect**
- контекст изображения: тип **HDC**;
- перо для рисования линий и фигур: тип **HPEN**;
- кисть для закраски ограниченных областей: тип **HBRUSH**;
- фон и цвет символов всего окна: константа **system("color F0")**;
- фон и цвет символов отображаемой строки текста: функции **SetBkColor**, **SetTextColor**.

Подключаемые **заголовочные файлы** для работы с графикой в окне консольного приложения: **windows.h** и **windowsx.h**

1.3.1.2. Работа с цветом

Для задания цветовых характеристик окна в глобальной области программы определяем следующую константу:

`const int NotUsed = system("color F0");` где значение color: F – белый цвет фона, 0 – черный цвет символов.

Системные цвета определяются так:

0= Black	1= Blue	2= Green	3= Aqua
4= Red	5= Purple	6= Yellow	7= White
8= Gray	9= Light Blue	A= Light Green	B= Light Aqua
C= Light Red	D= Light Purple	E= Light Yellow	F= Bright White

Задание цветовых характеристик отображаемой строки текста с использованием RGB-представления цвета приведено в табл. 8.

Таблица 8

Цветовые характеристики текста

RGB	Название	Демонстрация
(255;255;255)	Белый	
(0;0;0)	Черный	
(255;0;0)	Красный	
(128;0;0)	Темно-красный	
(0;255;0)	Зеленый	
(0;128;0)	Темно-зеленый	
(0;0;255)	Синий	
(0;0;128)	Темно-синий	
(0;255;255)	Голубой	
(0;128;128)	Темно-голубой	
(255;0;255)	Пурпурный	
(128;0;128)	Темно-пурпурный	
(255;255;0)	Желтый	
(128;128;0)	Темно-желтый	
(160;160;164)	Серый	
(128;128;128)	Темно-серый	
(192;192;192)	Светло-серый	

Для очистки прямоугольной области окна следует использовать функцию `InvalidateRect`:

```
bool InvalidateRect(
    HWND hwnd,           //дескриптор окна
    const RECT* lprec,    //координаты прямоугольника
    bool erase            //состояние очистки
)
```

где состояние очистки: **true** – очистить фон области, **false** – не очищать.

Для очистки всего окна можно использовать следующий код:
`InvalidateRect ( hWnd, NULL, 1 );`

1.3.1.3. Рисование фигур

Эллипс:

```
Ellipse(hdc, rt.leftr, rt.top, rt.right, rt.bottom);
```

где

`RECT rt` — прямоугольник, куда вписан эллипс;
`rt.leftr, rt.top` — координаты левого верхнего угла;
`rt.right, rt.bottom` — координаты правого нижнего угла.

Окружность:

```
int cx = 400, cy = 100, radius = 50;  
Ellipse(hdc, cx - radius, cy - radius, cx + radius, cy + radius);
```

где

`cx, cy` — координаты центра окружности;
`radius` — радиус окружности.

Треугольник:

```
POINT points[] = { {150, 150}, {80, 250}, {220, 250} };  
Polygon (hdc, points, 3);
```

где

`points` — массив координат вершин треугольника.

Прямоугольник:

```
Rectangle (hdc, 400, 100, 500, 250);
```

где

400, 100 — координаты левого верхнего угла,
500, 250 — координаты правого нижнего угла.

Координаты углов можно задать через параметры `left`,
`top`, `right`, `bottom` переменной типа `RECT`.

Линия:

```
POINT bp;  
MoveToEx(hdc, 150, 150, &bp); //устанавливаем начальную точку линии  
LineTo(hdc, 500, 250);        //рисует линию из начальной точки в  
                               //точку с указанными координатами
```

Пример 9. Рисование фигур в окне консольного приложения

```
#include <iostream>
#include <windows.h>
#include <windowsx.h>
//#include <math.h>
#include <string>
using namespace std;

const int NotUsed = system("color F0");

int main() {
    HWND hwnd = GetConsoleWindow(); //получаем идентификатор окна
    HDC hdc = GetDC(hwnd);           //получаем контекст отображения
    RECT rt;                          //RECT - прямоугольник
    GetClientRect(hwnd, &rt);         //получаем размер окна
                                     //создаем красное перо
    HPEN hRedPen = CreatePen(PS_SOLID, 2, RGB(255, 0, 0));

    //создаем синее перо
    HPEN hBluePen = CreatePen(PS_SOLID, 2, RGB(0, 0, 255));

    //создаем зеленую кисть
    HBRUSH hGreenBrush = CreateSolidBrush(RGB(0, 255, 0));

    //создаем голубую кисть
    HBRUSH hLightBlue = CreateSolidBrush(RGB(0, 255, 255));

    //создаем желтую кисть
    HBRUSH hYellowBrush = CreateSolidBrush(RGB(255, 255, 0));

    InvalidateRect(hwnd, NULL, 1);    //очистить окно

    cout << "Console Window" << endl; //выводим строку

    Sleep(1200);

    //назначаем перо и кисть для рисования
    SelectObject(hdc, hRedPen); SelectObject(hdc, hGreenBrush);
    //рисует эллипс
    Ellipse(hdc, 100, 200, rt.right - 300, rt.bottom - 100);
    Sleep(1200); //задержка процесса выполнения 1200 мсек

    SelectObject(hdc, hLightBlue); //назначаем голубую кисть
    int cx = 400, cy = 300, radius = 50;
    //рисует окружность
    Ellipse(hdc, cx - radius, cy - radius, cx + radius, cy +
                                                    radius);
}
```

```

Sleep(1200);
//массив координат вершин треугольника
POINT points[] = { {250, 250}, {100, 350}, {320, 350} };
Polygon(hdc, points, 3); //рисуем треугольник
Sleep(1200);

SelectObject(hdc, hYellowBrush); //назначаем желтую кисть
Rectangle(hdc, 400, 300, 500, 450); //рисуем прямоугольник
Sleep(1200);

POINT bp;
SelectObject(hdc, hBluePen); //назначаем синее перо
MoveToEx(hdc, 250, 250, &bp); //устанавливаем начальную точку
//линии
LineTo(hdc, 500, 450); //рисуем линию из начальной точки
//в точку с указанными координатами

Sleep(1200);

SetBkColor(hdc, RGB(255, 255, 255)); //устанавливаем цвет фона
SetTextColor(hdc, RGB(0, 0, 0)); //устанавливаем цвет текста
//формируем строку с текстом и значениями координат окна
//(левый верхний угол, правый нижний угол)
string c = "Graphics in Console Window:" +
to_string(rt.left) + ", " + to_string(rt.top) + ", " +
to_string(rt.right) + ", " + to_string(rt.bottom);
// выводим строку графическими средствами
TextOutA(hdc, 100, 200, c.c_str(), c.length()+1);
Sleep(1200);

//установка позиции курсора в окне
//COORD p = {1, 1};
//SetConsoleCursorPosition (GetStdHandle(STD_OUTPUT_HANDLE),
p);

//освобождаем ресурсы
DeleteObject(hRedPen);
DeleteObject(hBluePen);
DeleteObject(hGreenBrush);
DeleteObject(hLightBlue);
DeleteObject(hYellowBrush);
DeletePen(hRedPen);
DeleteBrush(hGreenBrush);
//освобождаем контекст отображения
ReleaseDC(hwnd, hdc);

}

```

Скриншоты работы программы приведены на рис. 22 и 23.

Локальные		
Поиск (Ctrl+E) ← → Глубина поиска: 3 🔍 📄		
Имя	Значение	Тип
points	0x0077fb2c {{x=250 y=250}, {x=100 y=350}, {x=320 y=350}}	tagPOINT[3]
radius	50	int
rt	{LT(0, 0) RB(960, 480) [960 x 480]}	tagRECT
left	0	long
top	0	long
right	960	long
bottom	480	long
Видимые Локальные Контрольные значения 1		

Рис. 22. Значения координат консольного окна

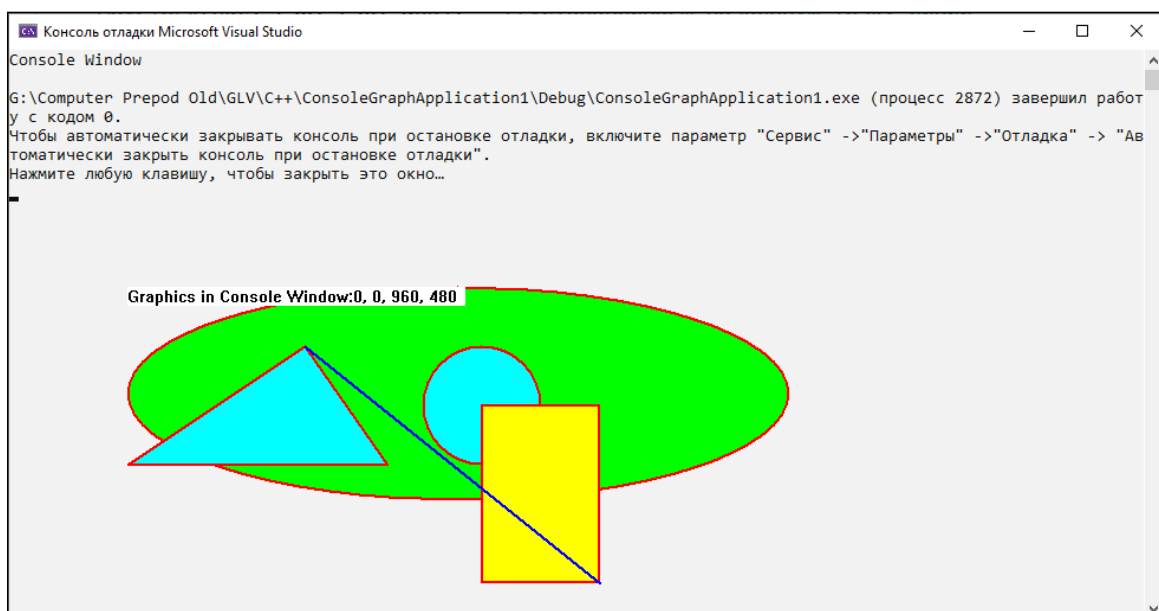


Рис. 23. Результат работы программы (Пример 9)

Пример 10. Классы фигур. Рисование в окне консольного приложения

В примере показано отображение объектов класса **Circle** с различными значениями цвета кисти:

```
#include <iostream>
#include <windows.h>
#include <windowsx.h>
#include <vector>
using namespace std;
```

```

const int NotUsed = system("color F0");
class Figure {
public:
    Figure(int _x, int _y, COLORREF c) : x(_x), y(_y), color(c) {
        hwnd = GetConsoleWindow(); //получаем идентификатор окна
        hdc = GetDC(hwnd);          //получаем контекст отображения
        GetClientRect(hwnd, &rt); //получаем прямоугольник окна
    }
    virtual void print() const {
        cout << x << ", " << y << ", ";
    }
    ~Figure() { ReleaseDC(hwnd, hdc); }
protected:
    int x, y;
    HWND hwnd;
    HDC hdc;
    RECT rt; //rt - прямоугольник окна
    COLORREF color;
};

class Circle : public Figure {
public:
    Circle(int x, int y, int a, COLORREF brush) : Figure(x, y,
                                                         brush), radius(a){}

    void print() const override {
        //выводим информацию об окружности
        cout << " Circle: "; Figure::print(); cout<<radius<<endl;
        //рисует окружность
        HPEN pen = CreatePen(PS_SOLID, 2, RGB(0, 0, 255));
        HBRUSH brush = CreateSolidBrush(color);
        SelectObject(hdc, pen);
        SelectObject(hdc, brush);
        Ellipse(hdc, x-radius, y-radius, x+radius, y+radius);
        //освобождаем ресурсы
        DeleteObject(pen);
        DeleteObject(brush);
    }
private:
    int radius;
};

int main() {
    SetConsoleTitle((LPCWSTR)L"22VP1_1");

    Sleep(1200);
    //устанавливаем позицию курсора в окне выводим текст
    COORD p = { 1, 1 };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), p);
}

```

```

cout << "Figures:" << endl;

//определяем vector для хранения фигур
vector<Figure*> v;
//добавляем окружность с зеленой кистью
v.push_back(new Circle(300, 100, 80, RGB(0, 255, 0)));
//добавляем окружность с желтой кистью
v.push_back(new Circle(350, 150, 50, RGB(255, 255, 0)));
//рисует фигуры
for (auto i : v) {
    i->print();
}

Sleep(3600);
p = { 1, 15 };
SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), p);
}

```

Скриншот работы программы приведен на рис. 24.

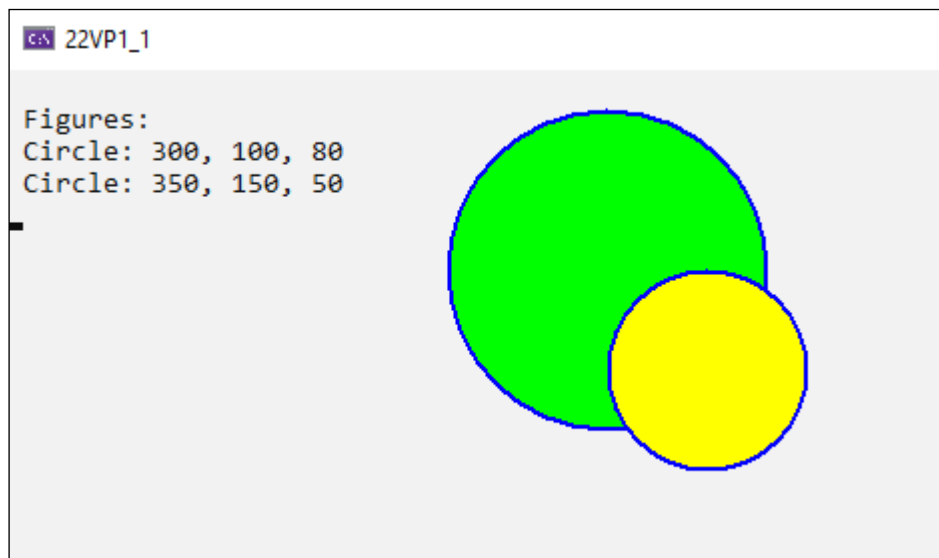


Рис. 24. Результаты работы программы (Пример 10)

1.3.2. Обработка исключительных ситуаций в курсовом проекте

Для обработки значений параметров геометрических фигур, которые приводят к неправильному отображению, в код приложения следует добавить классы исключений и операторы вывода с описанием исключительной ситуации.

Рассмотрим обработку исключительной ситуации нарушения левой границы окна консольного приложения при выводе изображения окружности:

1. Добавим в заголовочный файл **Figure.h** (пример 10) объявление класса **Border** для обработки исключения «Нарушение границы окна консольного приложения»:

```
class Figure {
public:
    Figure(int _x, int _y, COLORREF c) : x(_x), y(_y), color(c) {
        hwnd = GetConsoleWindow(); //получаем идентификатор окна
        hdc = GetDC(hwnd);          //получаем контекст отображения
        GetClientRect(hwnd, &rt);   //получаем прямоугольник окна
    }
    virtual void print() const {
        cout << x << ", " << y << ", ";
    }
    ~Figure() { ReleaseDC(hwnd, hdc); }
    class Border {}; //класс исключения
protected:
    int x, y;
    HWND hwnd;
    HDC hdc;
    RECT rt; //rt - прямоугольник окна
    COLORREF color;
};
```

После изменений класс **Figure** на диаграмме классов будет выглядеть следующим образом (рис. 25)

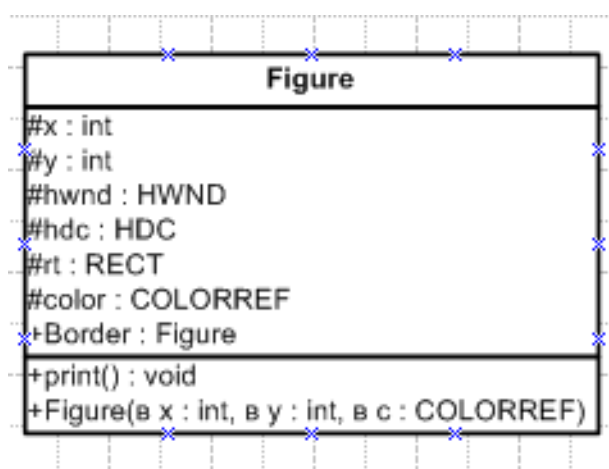


Рис. 25. Класс **Figure**

2. Добавим операторы, которые проверяют нарушение левой границы окна и генерируют исключение. Сделаем это на примере класса **Circle**:

```
void print() const override {
    //выводим информацию об окружности
    cout << " Circle: "; Figure::print(); cout << radius << endl;
    if (x - radius <= rt.left)
        throw Border();

    //рисует окружность
    HPEN pen = CreatePen(PS_SOLID, 2, RGB(0, 0, 255));
    HBRUSH brush = CreateSolidBrush(color);
    SelectObject(hdc, pen);
    SelectObject(hdc, brush);
    Ellipse(hdc, x - radius, y - radius, x + radius, y + radius);
    //освобождаем ресурсы
    DeleteObject(pen);
    DeleteObject(brush);
}
```

В сценарий варианта использования «Показать окружность» необходимо внести условие генерации исключения (табл. 9).

Таблица 9

Сценарий «Показать окружность»

Наименование: Показать окружность
ID: 10
Краткое описание: система рисует изображение окружности с заданными координатами центра (x, y) и радиусом (r)
Действующие лица: пользователь
Предусловие: окружность создана
Основной поток: - Пользователь инициирует рисование окружности с координатами центра (x, y) и радиусом (r) - Если условие ($x - r \leq$ левая граница окна) выполняется: - Система генерирует исключение - Иначе: - Система рисует изображение окружности с заданными координатами центра (x, y) и радиусом (r)
Постусловие: окружность нарисована

3. Добавим в главную программу операторы, которые будут обрабатывать исключения:

```
int main() {
    SetConsoleTitle((LPCWSTR)L"22VP1_1");

    Sleep(1200);
    //устанавливаем позицию курсора в окне
    COORD p = { 1, 1 };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), p);
    cout << "Figures:" << endl;

    vector<Figure*> v;
    try {
        v.push_back(new Circle(300, 100, 80, RGB(0, 255, 0)));
        v.push_back(new Circle(350, 150, 50, RGB(255, 255, 0)));
        for (auto i : v) {
            i->print();
        }
    }
    catch (Figure::Border) {
        cout << "breaking Window Border" << endl;
    }

    Sleep(3600);
    p = { 1, 15 };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), p);
}
```

При заданных данных Circle(300, 100, 80, RGB(0, 255, 0)) и Circle (350, 150, 50, RGB(255, 255, 0)) исключения не возникают – фигуры помещаются в окне (рис. 26).

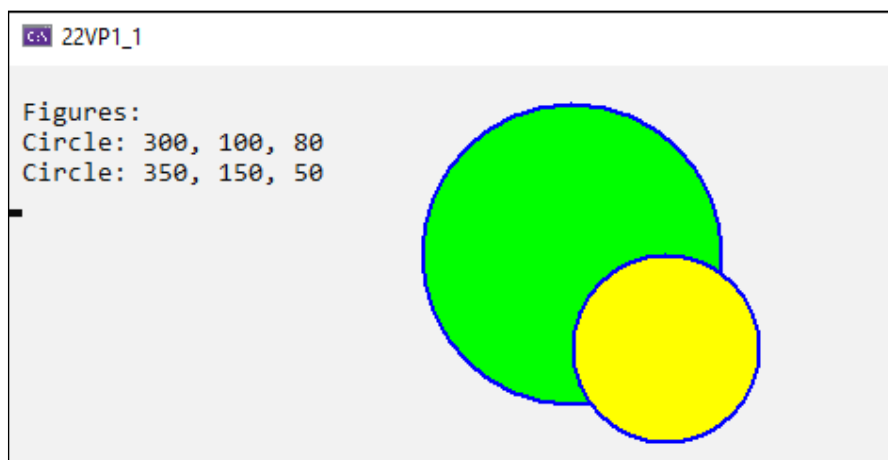


Рис. 26. Нет исключения

Изменим значения координат для окружности Circle(350, 150, 450, RGB(255, 255, 0)). В этом случае окружность выходит за границу окна: $cx = 350$; $radius = 450$; $rt.left = 0$ и $x - radius < rt.left$ – окружность не отображается и выдается сообщение о нарушении границ (рис. 27).

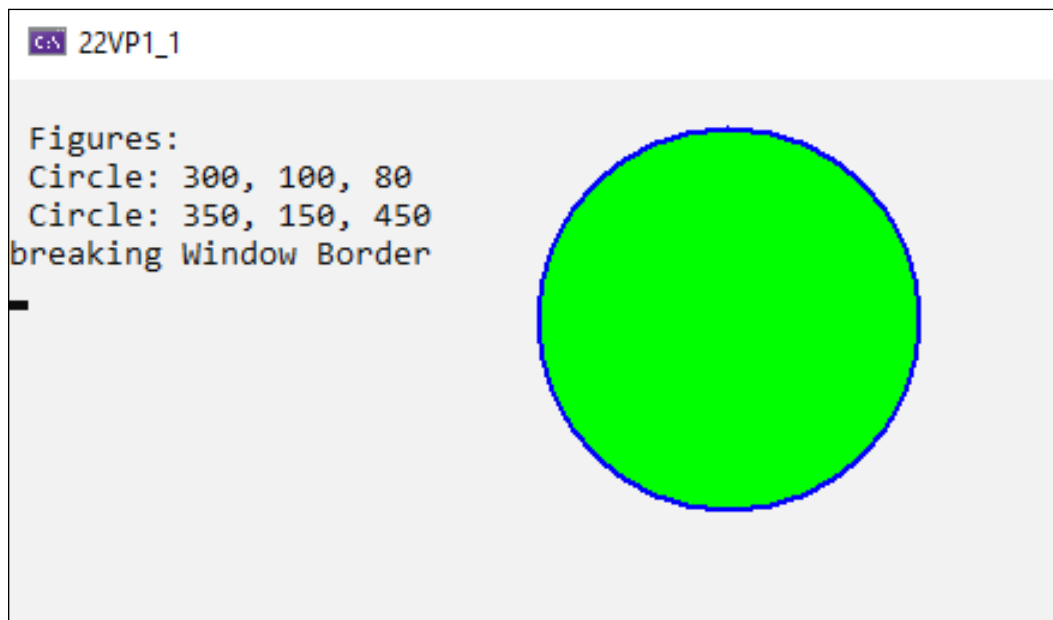


Рис. 27. Исключение есть

1.3.3. Рисование фигур с помощью платформы .NET Framework

Класс **Graphics** представляет поверхность рисования, объект **Graphics** используется для создания графических изображений. Таким образом, работа с графикой состоит из двух этапов:

- создание **Graphics** объекта;
- использование **Graphics** объекта для рисования линий и фигур, отображения текста или отображения изображений и управления ими.

Базовые классы для создания графических изображений:

- **Pen** – класс, используемый для рисования линий, структурирования фигур или визуализации других геометрических представлений;
- **Brush** – класс, используемый для заливки областей графических объектов, таких как заполненные фигуры, изображения или текст;
- **Font** – класс содержит описание фигур, используемых при отрисовке текста;
- **Color** – структура представляет различные отображаемые цвета.

Пример 11. Классы фигур. Рисование с помощью платформы .NET Framework

Главная программа (**MyForm.cpp**) и структура проекта показаны на рис. 28.

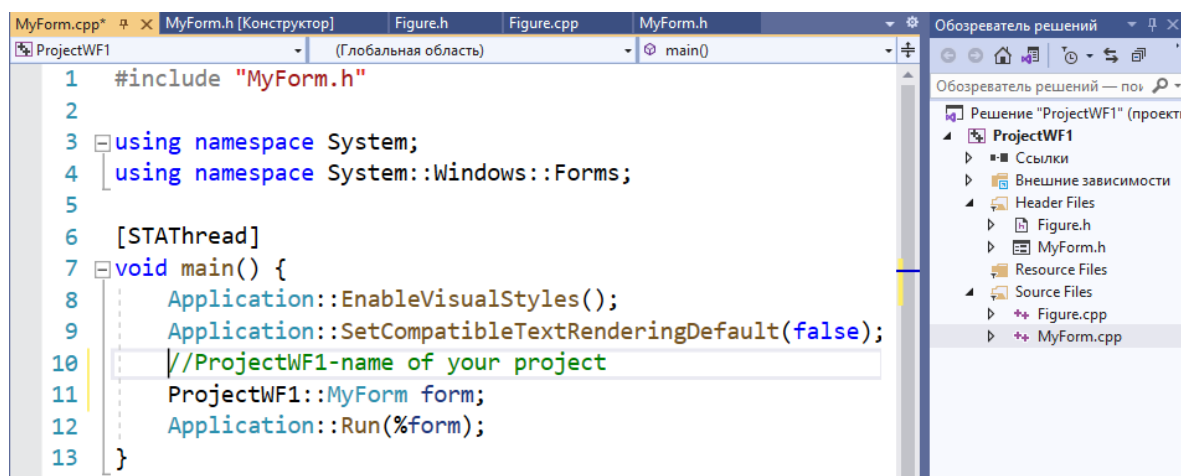


Рис. 28. Главная программа и структура проекта

Для создания формы приложения используется Конструктор формы – **MyForm.h [Конструктор]** (рис. 29).

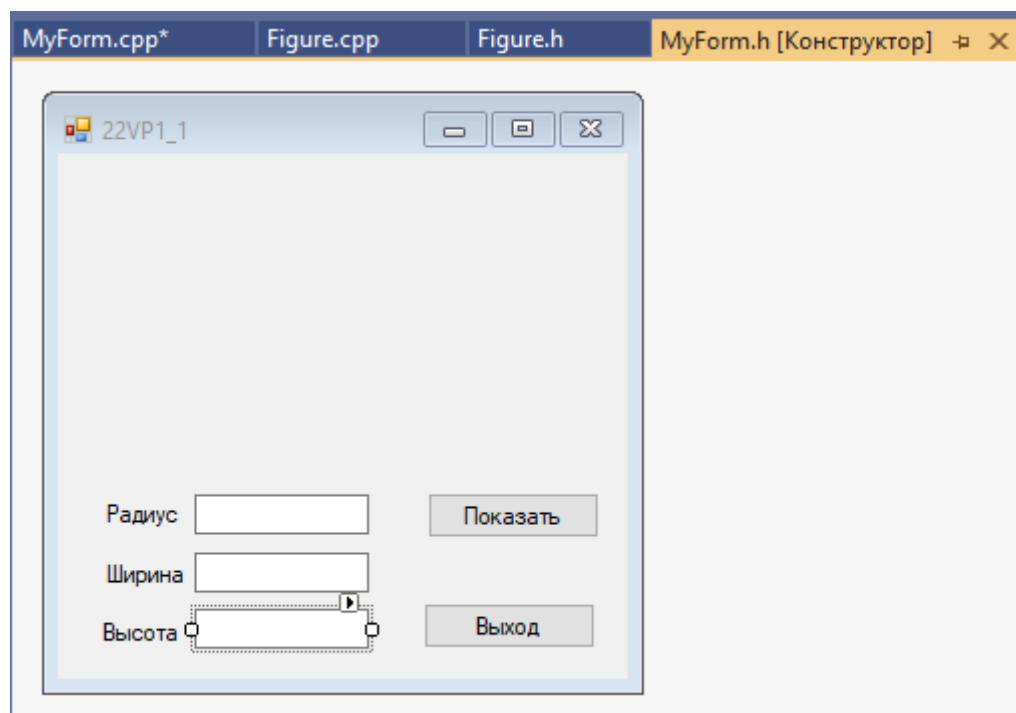


Рис. 29. Форма приложения

Создание формы осуществляется из элементов Панели элементов и настройки их свойств. Окно свойств (иконка  на панели инструментов) показано на примере настройки свойств формы **MyForm** (рис. 30) и добавляемых на форму элементов **label1** (рис. 31), **textBox1** (рис. 32), **button1** (рис. 33). Свойства остальных элементов настраиваются аналогично.

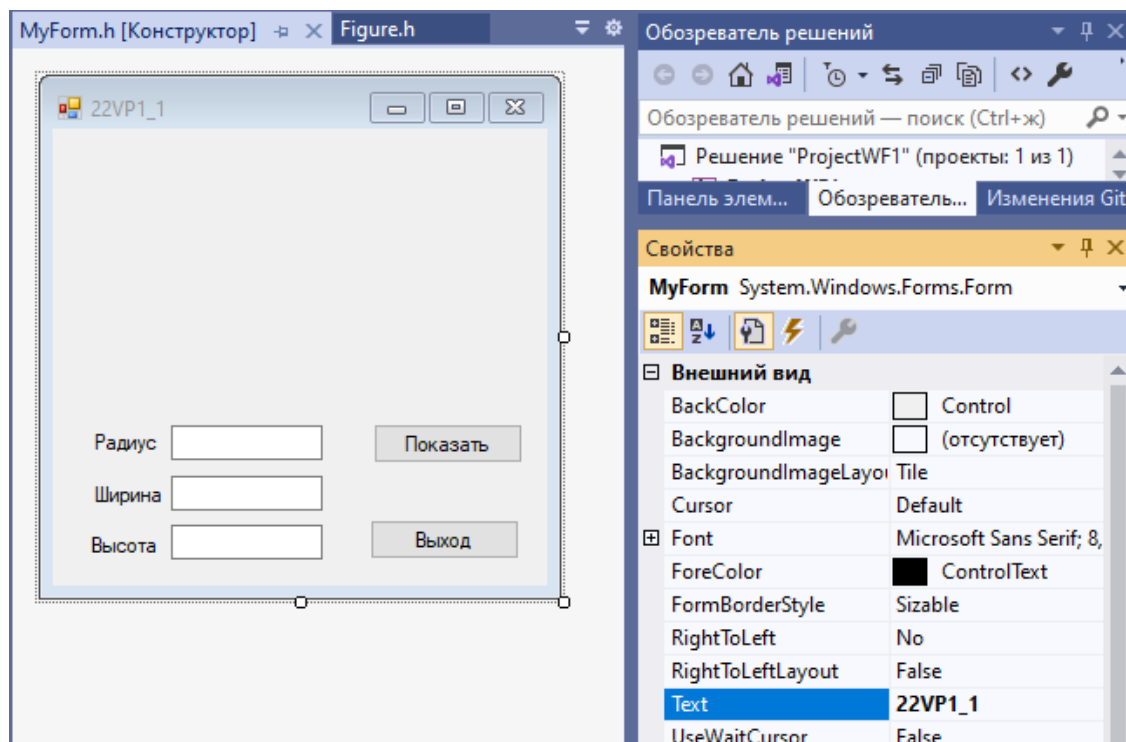


Рис. 30. Свойства формы **MyForm**

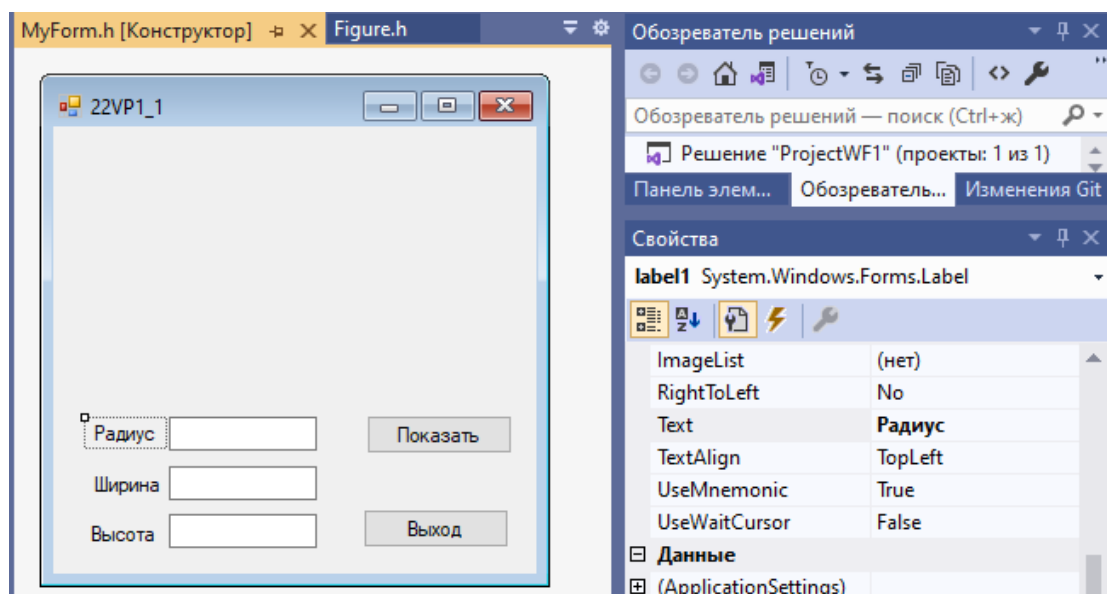


Рис. 31. Свойства надписи **label1**

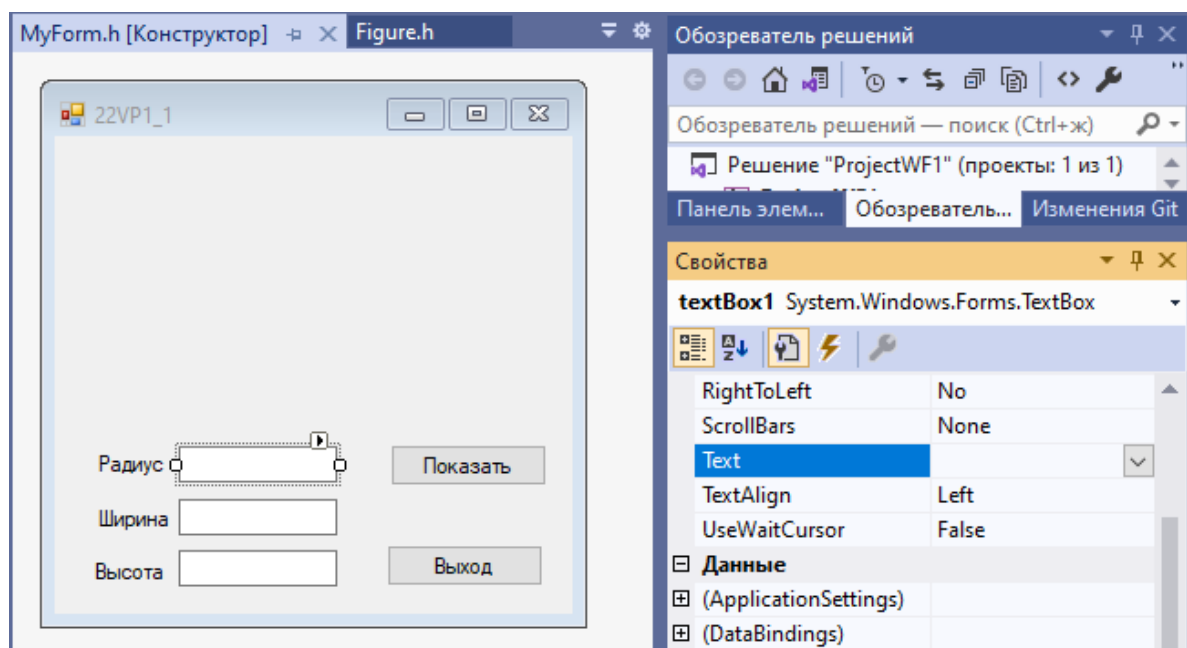


Рис. 32. Свойства поля ввода **textBox1**

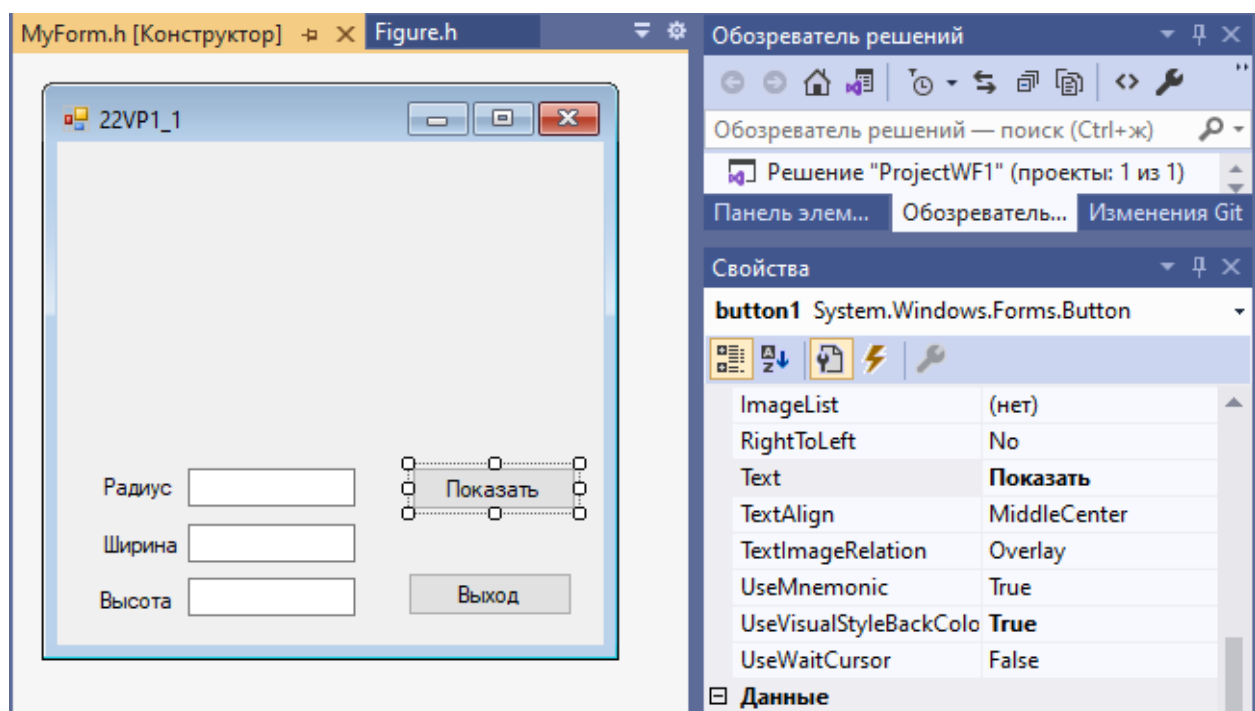


Рис. 33. Свойства кнопки **button1**

Добавим обработку события «Нажатие кнопки» для кнопки **button1**. В окне свойств открываем задание обработчиков событий (иконка ), выбираем событие **Click** и связываем с ним функцию **OnButtonClick** (рис. 34).

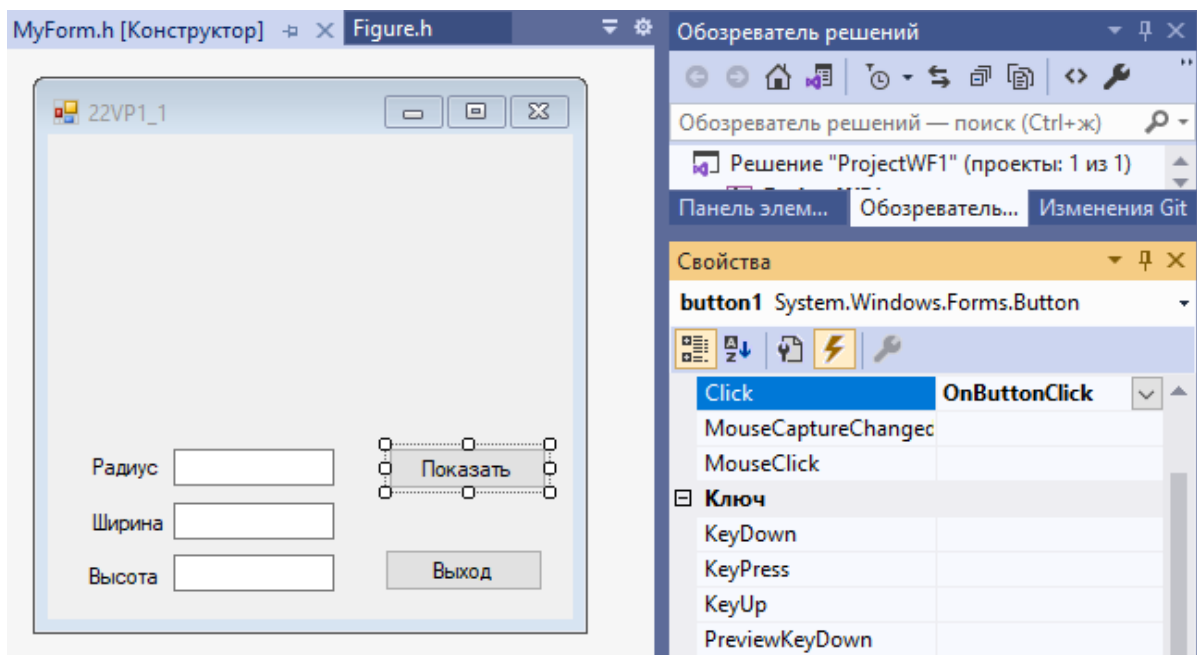


Рис. 34. Задание обработчика для события нажатия на кнопку

После этого в коде будет автоматически сформирована пустая функция `OnButtonClick`. В реализацию этой функции помещаем код, связанный с созданием и отрисовкой фигур:

MyForm.cpp

```
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThread]
void main() {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    //ProjectWF1-name of your project
    ProjectWF1::MyForm form;
    Application::Run(%form);
}
```

Figure.h

```
#pragma once
using namespace System::Drawing;

class Figure {
public:
```

```

    Figure() : x(0), y(0) {}
    virtual void draw(Graphics^ g) = 0;
protected:
    int x, y;
};

class MyRectangle : public Figure {
public:
    MyRectangle(int a, int b) : height(a), width(b) {}
    void draw(Graphics^ g) override;
private:
    int height, width;
};

class MyCircle : public Figure {
public:
    MyCircle(int a) : radius(a) {}
    void draw(Graphics^ g) override;
private:
    int radius;
};

```

Figure.cpp

```

#include "Figure.h"
using namespace System::Drawing;

void MyRectangle::draw(Graphics^ g) {
    Pen^ bluePen = gcnew Pen(Color::Blue);
    bluePen->Width = 5;
    //Показать прямоугольник
    g->DrawRectangle(bluePen, x, y, height, width);
}

void MyCircle::draw(Graphics^ g) {
    Pen^ redPen = gcnew Pen(Color::Red);
    redPen->Width = 5;
    //Создать прямоугольник для эллипса
    Rectangle rect = Rectangle(x, y, radius, radius);
    //Показать эллипс
    g->DrawEllipse(redPen, rect);
}

```

MyForm.h

```

#pragma once
#include <string>
#include "Figure.h"

```

```

#using <system.drawing.dll>

namespace ProjectWF1 {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for MyForm
    /// </summary>
    public ref class MyForm : public System::Windows::Forms::Form
    {
    public:
        MyForm(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~MyForm()
        {
            if (components)
            {
                delete components;
            }
        }

    private: System::Windows::Forms::TextBox^ textBox1;
    private: System::Windows::Forms::Button^ button1;
    private: System::Windows::Forms::Button^ button2;
    private: System::Windows::Forms::TextBox^ textBox2;
    private: System::Windows::Forms::TextBox^ textBox3;
    private: System::Windows::Forms::Label^ label1;
    private: System::Windows::Forms::Label^ label2;
    private: System::Windows::Forms::Label^ label3;
    protected:

```

```

private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        this->textBox1 = (gcnew System::Windows::Forms::Text-
            Box());
        this->button1 = (gcnew System::Windows::Forms::But-
            ton());
        this->button2 = (gcnew System::Windows::Forms::But-
            ton());
        this->textBox2 = (gcnew System::Windows::Forms::Text-
            Box());
        this->textBox3 = (gcnew System::Windows::Forms::Text-
            Box());
        this->label1 = (gcnew System::Windows::Forms::La-
            bel());
        this->label2 = (gcnew System::Windows::Forms::La-
            bel());
        this->label3 = (gcnew System::Windows::Forms::La-
            bel());
        this->SuspendLayout();

        //Инициализация компонент textBox1, textBox2, textBox3,
        //button1, button2, label1, label2, label3
        . . .

        //
        //MyForm
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(6,
            13);
        this->AutoScaleMode = System::Windows::Forms::Au-
            toScaleMode::Font;
        this->ClientSize = System::Drawing::Size(284, 262);
        this->Controls->Add(this->label3);
        this->Controls->Add(this->label2);
        this->Controls->Add(this->label1);
    }

```

```

        this->Controls->Add(this->textBox3);
        this->Controls->Add(this->textBox2);
        this->Controls->Add(this->button2);
        this->Controls->Add(this->button1);
        this->Controls->Add(this->textBox1);
        this->Name = L"MyForm";
        this->Text = L"22VP1_1";
        this->ResumeLayout(false);
        this->PerformLayout();
    }
#pragma endregion
    private: System::Void OnButtonClick(System::Object^ sender,
                                         System::EventArgs^ e) {
        Graphics^ h = CreateGraphics();
        h->Clear(Color::AntiqueWhite);

        Rectangle rect = Form::ClientRectangle;
        Pen^ greenPen = gcnew Pen(Color::Green);
        greenPen->Width = 5;
        h->DrawLine(greenPen, 0, 0, rect.Width, rect.Height);

        //пример обработки строк
        //int radius = System::Convert::ToInt32(textBox1->Text);
        //std::string s = std::to_string(radius);
        //String^ str = gcnew System::String(s.c_str());

        int radius = System::Convert::ToInt32(textBox1->Text);
        MyCircle mycircle(radius);
        mycircle.draw(h);

        int width = System::Convert::ToInt32(textBox2->Text);
        int height = System::Convert::ToInt32(textBox3->Text);
        MyRectangle myrectangle(height, width);
        myrectangle.draw(h);
    }

    private: System::Void OnClickCloseButton(System::Object^
sender,
                                         System::EventArgs^ e) {
        Close();
    }
};
}

```

Результат работы приложения с заданными значениями параметров фигур приведен на рис. 35.

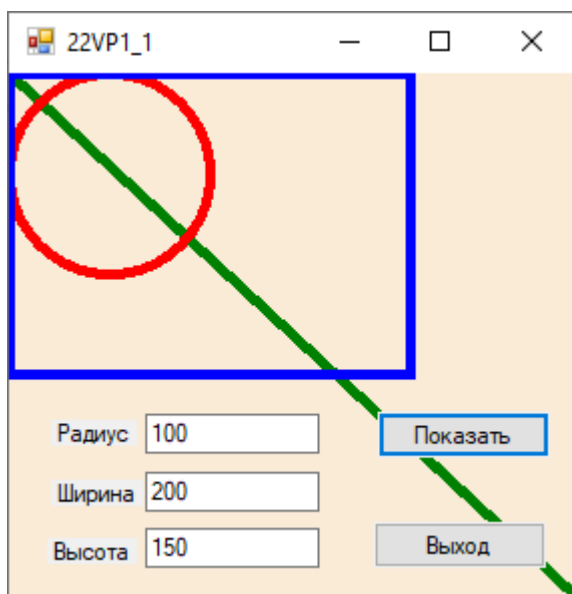


Рис. 35. Результаты работы (Пример 11)

1.3.4. Рекомендации к реализации программных средств

Для реализации программных средств курсового проекта следует:

- хорошо изучить диаграмму классов;
- реализовать многофайловое приложение на языке C++, в котором каждому классу соответствует заголовочный файл и файл реализации;
- правильно реализовать спроектированные отношения между классами (наследование, абстракция, агрегация, композиция);
- в обработку исключительных ситуаций добавить вывод параметров, значения которые вызвали исключение;
- в описание раздела включить диаграмму компонентов и таблицу их описания;
- диаграмму компонентов, кроме рисунка в разделе «Реализация», необходимо выполнить в формате А3 и поместить в графическую часть пояснительной записки (приложение Б);
- программный код приложения поместить в приложение А.

1.4. Тестирование

Тестирование – деятельность, выполняемая для оценки и улучшения качества программного обеспечения. Эта деятельность базируется на обнаружении дефектов и проблем в программных системах.

Модель тестирования описывает, как с помощью тестов проверяются исполняемые компоненты модели реализации. Модель тестирования включает:

- набор тестовых примеров (тестов);
- классы и функции тестирования;
- тестовые компоненты (созданные для автоматизации тестирования).

В рамках курсового проекта основной методологией тестирования обычно является функциональное тестирование.

Функциональное тестирование – это тестирование программных средств с целью проверки правильности реализации функциональных требований. Рекомендуется использовать следующие методы функционального тестирования:

- классы эквивалентности;
- метод граничных значений.

Класс эквивалентности – набор данных, обрабатываемый одинаковым образом и приводящий к одинаковому результату. В курсовом проекте следует выделить:

- правильные классы эквивалентности – координаты центра, радиус и другие параметры геометрических фигур, при задании которых фигуры полностью отображаются в окне консольного приложения;
- неправильные классы эквивалентности – координаты центра, радиус и другие параметры геометрических фигур, при задании которых фигуры отображаются не полностью или не отображаются в окне консольного приложения.

Граничное условие – значение, находящееся на границе классов эквивалентности.

План функционального тестирования формируется на основе вариантов использования. План должен включать варианты использования с описанием тестов, соответствующих классам эквивалентности для этого варианта, и результатами выполнения тестов. Пример плана тестирования для тестирования вариантов использования **Создать окружность** и **Показать окружность** примера 10 приведен в табл. 10.

План тестирования

Вариант использования	Тест	Результат
Создать окружность	Тест 1 Координаты центра: 300, 100 Радиус: 80 Цвет заполнения: RGB(0, 255, 0)	Тест пройден. Объект создан
	Тест 2 Координаты центра: 350, 150 Радиус: 50 Цвет заполнения: RGB(255, 255, 0)	Тест пройден. Объект создан
	Тест 3 ...	...
Показать окружность	Тест 1 Координаты центра: 300, 100 Радиус: 80 Цвет заполнения: RGB(0, 255, 0)	Тест пройден. Окружность нарисована правильно (рис. 26)
	Тест 2 Координаты центра: 350, 150 Радиус: 50 Цвет заполнения: RGB(255, 255, 0)	Тест пройден. Окружность нарисована правильно (рис. 26)
	Тест 3 Координаты центра: 350, 150 Радиус: 450 Цвет заполнения: RGB(255, 255, 0)	Тест пройден. Окружность не нарисована. Сгенерировано и обработано исключение (рис. 27).
...	...	...

Для разработки плана тестирования следует:

- разбить область данных на классы эквивалентности;
- составить тесты (значения параметров геометрических фигур) для каждого класса эквивалентности. Тестов должно быть достаточно для того, чтобы показать возможные варианты фигур (например, для окружности тесты с различными значениями радиуса и координат центров);
- составить план тестирования;
- добавить в код приложения обработку исключительных ситуаций (неправильных классов эквивалентности);
- выполнить тесты;
- результаты тестирования подтвердить с помощью скриншотов.

Скриншоты следует использовать «экономно»: не надо все тесты сопровождать скриншотами. Количество скриншотов должно быть ограничено (не более 5). Можно приводить один скриншот для нескольких тестов, например, для тестов «Показать окружность» и «Показать прямоугольник» приводить скриншот, на котором показаны обе фигуры.

Скриншоты следует форматировать (удалять пустые области) перед их размещением в пояснительной записке (формат размещения – «в тексте»).

2. СТРУКТУРА И ОФОРМЛЕНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

2.1. Структура пояснительной записки

Структура пояснительной записки включает:

- титульный лист;
- техническое задание;
- содержание;
- введение (не нумеруется);
- разделы разработки программных средств;
- заключение (не нумеруется);
- список использованных источников (не нумеруется);
- приложения не нумеруются; приложение А – исходный код программы; приложение Б – графическая часть, **2 листа формата А3** с диаграммами UML.

Рекомендуемый объем курсового проекта – **20–30 страниц** печатного текста без учета приложений.

2.1.1. Титульный лист и техническое задание

Тема задания на титульном листе, в техническом задании и в первом разделе содержания (заголовок первого уровня) совпадает с темой в ЭИОС!

Код проекта в примере титульного листа:

ПГУ 09.03.04 – 02КП221. 01 ПЗ

где **09.03.04** – направление «Программная инженерия»; **02** – второй семестр; **КП** – курсовой проект; **22** – год поступления; **1** – номер группы (22ВП1); **01** – порядковый номер студента в списке группы (каждый ставит свой номер); **ПЗ** – пояснительная записка.

Пример оформления титульного листа и технического задания приведен далее.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Математическое обеспечение и применение ЭВМ»

Курсовой проект

по дисциплине «Программирование»
на тему «Разработка консольного приложения
с элементами графики: треугольники»

ПГУ 09.03.04 – 02КП221. 01 ПЗ
Направление подготовки – 09.03.04 «Программная инженерия»

Выполнил студент:

ФИО студента

Группа:

22ВП1

Руководитель:

к.т.н., доцент

Гурьянов Л. В.

Проект защищен с оценкой

Преподаватели

Дата защиты

Пенза 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

«УТВЕРЖДАЮ»

Зав. кафедрой МОиПЭВМ

_____ А. Ю. Козлов

ЗАДАНИЕ

на курсовой проект
по дисциплине «Программирование»
на тему «Разработка консольного приложения
с элементами графики: треугольники»

1. Студент гр.	22ВПИ	факультета ВТ	направления 09.03.04
ФИО студента			
2. Руководитель работы	Гурьянов Лев Вячеславович		
3. Время проектирования	с «15» февраля 2023 г.	по «30» мая 2023 г.	
4. Тема проекта: Разработка консольного приложения с элементами графики: треугольники			
5. Техническое задание на курсовую работу (назначение, технические требования)			
<u>Назначение:</u> создание и визуализация следующих типов фигур:			
треугольник, сложная фигура (треугольник, вписанный в треугольник)			
<u>Основные функции приложения:</u>			
а) создать геометрическую фигуру (треугольник, сложная фигура);			
б) показать/скрыть фигуру;			
в) переместить фигуру;			
...			
<u>Структура данных и программных средств:</u> список, базовый абстрактный класс,			
<u>Технология разработки:</u> ООП			

Язык программирования: C++

Среда исполнения: консольное приложение Windows

6. Содержание работы

6.1. Пояснительная записка (перечень вопросов, подлежащих разработке, расчетов, обоснований, описаний)

- 1) Анализ предметной области
- 2) Анализ функциональных требований
- 3) Проектирование
- 4) Реализация
- 5) Тестирование
- 6) Оформление пояснительной записки

7. Календарный график по выполнению работы

Наименование этапов работы	Объем работы (%)	Срок выполнения	Подпись руководителя
1) Анализ предметной области и требований	10	27.02.2023	
2) Проектирование	20	20.03.2023	
3) Реализация	30	20.04.2023	
4) Тестирование	20	15.05.2023	
5) Оформление пояснительной записки	20	28.05.2023	

Дата выдачи задания «15» февраля 2023 г.

Руководитель курсового проекта _____

Задание к исполнению принял «15» февраля 2023 г.

Студент _____

2.1.2. Содержание и разделы разработки программных средств

Автоматически построенное содержание следует после технического задания. Содержание пояснительной записки должно быть следующим:

Введение.....

1. Название раздела совпадает с темой курсового проекта

1.1. Анализ предметной области

1.2. Анализ функциональных требований	
1.3. Проектирование	
1.4. Реализация	
1.5. Тестирование.....	
Заключение	
Список использованных источников.....	
Приложение А. Код приложения	
Приложение Б. Графическая часть	

Введение

Введение содержит краткое описание постановки задачи курсового проекта и используемых средств моделирования (язык, среда разработки) и средств разработки (язык, платформа, среда разработки).

Во введении следует перечислить основные этапы разработки (см. разделы разработки ПС) и указать содержимое приложений.

Разделы разработки программных средств

Разделы разработки ПС оформляются в пояснительной записке как подразделы второго уровня. Требования по содержанию разделов, связанных с разработкой ПО, следующие:

- **анализ предметной области:** рисунок из задания (геометрические фигуры), модель предметной области и ее краткое описание;
- **анализ функциональных требований:** диаграмма вариантов использования, сценарии вариантов использования (2–3 сценария для примера), диаграмма последовательности (если сложный сценарий и необходимо его пояснение);
- **проектирование программного обеспечения:** диаграмма классов с пояснением, краткая спецификация классов (назначение класса; члены класса, данные – название, тип; функции – название, параметры, назначение);
- **реализация:** диаграмма компонентов (файлы *.cpp, *.h, *.exe), таблица описания компонентов;
- **тестирование:** таблица «План тестирования» и скриншоты с примерами выполнения тестов.

Заключение

Заключение содержит развернутый текст о выполненных этапах разработки с указанием конкретных результатов каждого этапа (моделях, диаграммах, классах, компонентах, результатов тестирования).

2.2. Требования к оформлению пояснительной записки

2.2.1. Общие требования

Пояснительная записка выполняется на стандартных листах формата А4 с соблюдением следующих требований: поля: левое – 30 мм, правое – 15 мм, верхнее – 20 мм, нижнее – 20 мм; шрифт размером 13 или 14 пунктов, гарнитура Times New Roman; межстрочный интервал – полуторный; отступ – 1,25; выравнивание текста – по ширине.

Все страницы пояснительной записки должны быть пронумерованы последовательно арабскими цифрами. Номер страницы размещается в центре нижней части листа. Нумерация страниц должна быть сквозной от титульного листа (на титульном листе и задании на работу нумерация не ставится, но в общее количество листов включается) до последнего листа текста, включая иллюстративный материал (таблицы, графики, диаграммы и т.п.), расположенный внутри текста, а также приложения.

Заголовки разделов и подразделов пояснительной записки имеют абзацный отступ. Точка после номера последнего раздела/подраздела и в конце заголовка не ставится, например:

- 1 Разработка консольного приложения с элементами графики**
- 1.1 Анализ предметной области и функциональных требований**
- 1.1.1 Модель предметной области**

2.2.2. Требования к оформлению перечислений

Перечисления оформляются следующим образом, например:

а) семантика *агрегации*:

– агрегат может существовать как независимо от частей, так и вместе с ними;

– части могут существовать независимо от агрегата;

б) семантика композиции:

– одновременно части могут принадлежать только одному композиту – совместное владение частями невозможно;

– композит обладает исключительной ответственностью за все свои части – это означает, что он отвечает за их создание и уничтожение.

2.2.3. Требования к оформлению таблиц

Таблицы последовательно нумеруются арабскими цифрами в пределах всей работы и размещаются сразу после ссылки на них в тексте. Название таблиц размещается над левым верхним углом соответствующей таблицы, выравниваются по левому краю, шрифт Times New Roman, размер шрифта – от 10 до 14 (в зависимости от объема таблицы), межстрочный интервал 1, в параметрах абзаца интервал перед – 15 пт, интервал после – 10 пт.

Если требуется перенести таблицу на другую страницу, то ее «шапку» необходимо будет продублировать на каждой новой странице пояснительной записки, а прерывание таблицы внизу сопровождается отсутствием нижней границы у всей строки.

Если таблица размещается на двух страницах пояснительной записки, то ее заголовок на второй странице меняется на «Продолжение таблицы N», где N номер соответствующей таблицы. В случае, если таблица занимает более чем две страницы, то на последней странице с таблицей в заголовке указывается «Окончание таблицы N».

На все таблицы в пояснительной записке обязательно должны быть ссылки в тексте. Ссылаться на данные таблицы можно напрямую в тексте самого предложения или косвенно в круглых скобках с указанием порядкового номера соответствующей таблицы.

Например: «Описание компонентов приведено в таблице N.

Таблица N – Компоненты приложения

Компоненты	Назначение
Figure.h	Заголовочный файл класса Figure
Figure.cpp	Исходный файл класса Figure
Circle.h	Заголовочный файл класса Circle
Circle.cpp	Исходный файл класса Circle
Rectangle.h	Заголовочный файл класса Rectangle
Rectangle.cpp	Исходный файл класса Rectangle
windows.h	Системный заголовочный файл
Windowsx.h	Системный заголовочный файл
Project22vp1_1.cpp	Главная программа
Project22vp1_1.exe	Исполняемый файл приложения

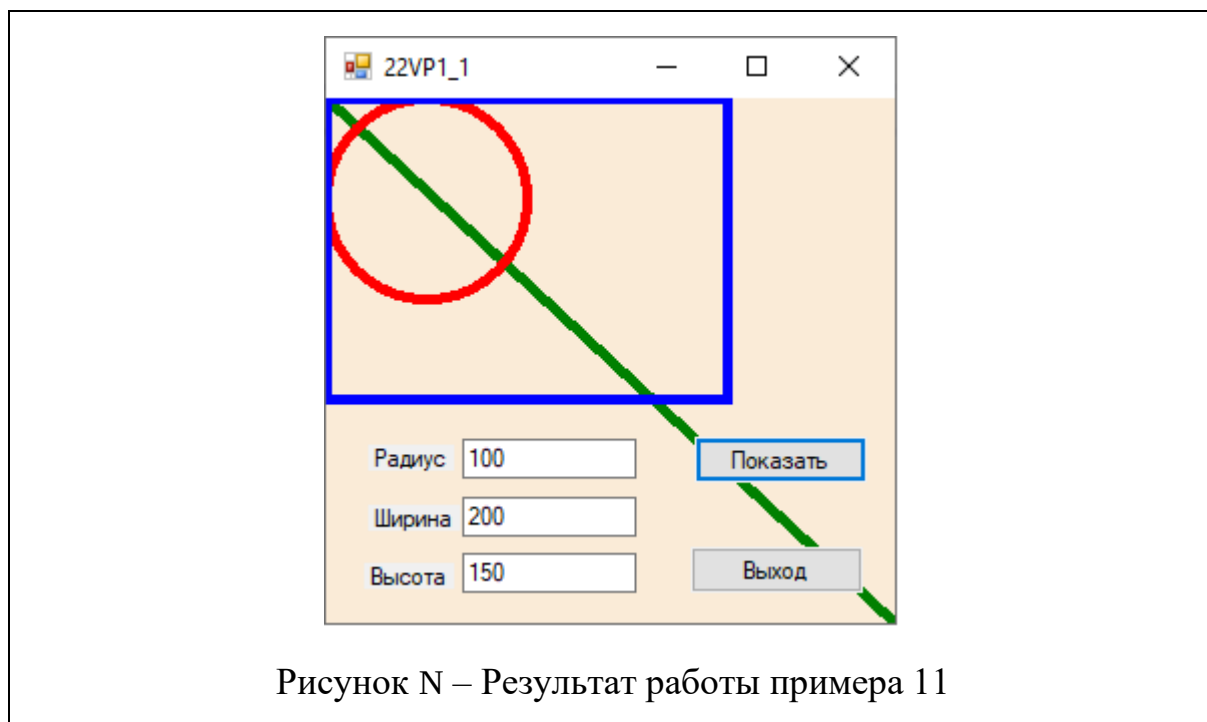
2.2.4. Требования к оформлению рисунков

Все иллюстрации, не относящиеся к таблицам (схемы, графики, диаграммы и т.д.), именуются рисунками.

Формат размещения рисунков – «в тексте».

Рисункам присваивается сквозная нумерация в пределах всей работы, за исключением иллюстраций приложений. Иллюстрации каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения.

Например: «Результаты работы примера 11 приведен на рисунке N».



Все рисунки должны иметь полные наименования. Номер и наименование рисунка записываются через тире в строчку под его изображением без закрывающей точки. Подрисуночная надпись начинается без абзацного отступа (красной строки), выравнивается по центру и оформляется шрифтом Times New Roman, размер шрифта – 14, межстрочный интервал 1, в параметрах абзаца отступ слева – 0 см, отступ справа – 0 см, интервал перед – 10 пт, интервал после 15 пт. Ссылка на рисунок должна стоять перед самым рисунком, на который она ссылается. Ссылаться на рисунки можно напрямую в тексте самого предложения или косвенно в круглых скобках с указанием порядкового номера соответствующего рисунка.

Например: «Результат работы приложения с заданными значениями параметров фигур приведен на рисунке».

2.2.5. Требования к оформлению используемых источников

Использованные в процессе работы специальные литературные источники указываются в конце пояснительной записки перед приложениями. Список использованных источников входит в основной объем работы. На каждый литературный источник в тексте работы обязательно должна быть хотя бы одна ссылка. Ссылка – номер источника – задается в квадратных скобках. Список литературы составляется в порядке упоминания литературных источников в работе.

Пример ссылки на книгу:

1. Арлоу Дж., Нейштадт А. UML2 и Унифицированный процесс. Практический объектно ориентированный анализ и проектирование. – СПб. : Символ-Плюс, 2007. – 624 с.

2. Павловская Т. А. С/С++. Программирование на языке высокого уровня. – СПб. : Питер, 2003. – 461 с.

Пример описания интернет-ресурса:

Руководство по языку программирования С++. – URL: <https://metanit.com/cpp/tutorial/> (дата обращения: 02.06.2022).

2.2.6. Требования к оформлению приложений

Курсовой проект содержит два приложения: А – код программы, Б – графическая часть. Название приложения оформляется на отдельном листе по центру. На следующих листах располагается содержимое приложения.

Название для приложения А:

Приложение А
КОД ПРОГРАММЫ
(обязательное)

Название для приложения Б:

Приложение Б
ГРАФИЧЕСКАЯ ЧАСТЬ
(обязательное)

Примеры оформления приложений приведены в приложении А и Б.

3. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ ПО КУРСОВОМУ ПРОЕКТУ

3.1. Предметная область

Предметная область курсового проекта приведена на рис. 36.

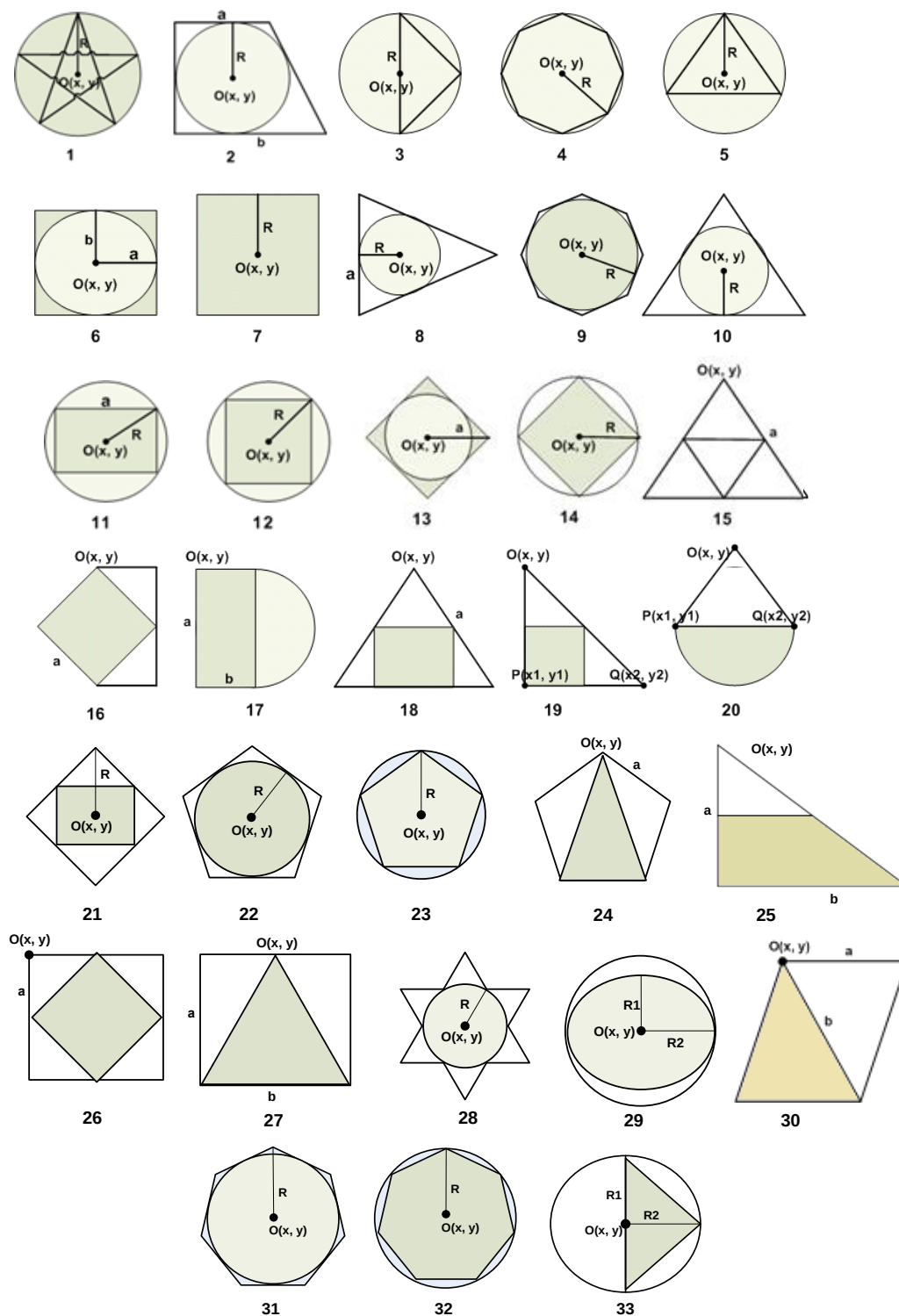


Рис. 36. Предметная область курсового проектирования

3.2. Функциональные требования и требования к реализации

Функциональные требования к программным средствам курсового проекта следующее:

- создать геометрическую фигуру (фигуры из задания, сложная фигура);
- показать (нарисовать) фигуру;
- спрятать (удалить с экрана) фигуру;
- переместить фигуру;
- добавить фигуру в контейнер хранения;
- показать фигуры из контейнера;
- удалить контейнер фигур.

Конкретные фигуры выбираются по варианту из предметной области задания (см. рис. 36).

Конкретный контейнер и структура программных средств (абстракция, агрегация, композиция) выбираются по варианту из табл. 11.

Таблица 11

Задания на курсовой проект

№	Структура данных хранения фигур							Структура программных средств			
	Список	Динамический массив	Стек		Множество	Очередь		Фигура		Сложная фигура	
			Динамический массив	Список		Двусторонняя очередь	Список	Базовый класс	Базовый абстрактный	Агрегация	Композиция
1	2	3	4	5	6	7	8	9	10	11	12
1	+							+		+	
2		+						+		+	
3			+					+		+	
4				+				+		+	
5					+			+		+	
6						+			+		+
7							+		+		+
8						+			+		+
9					+				+		+
10				+					+		+
11			+						+	+	

1	2	3	4	5	6	7	8	9	10	11	12
12		+							+	+	
13	+								+	+	
14		+							+	+	
15			+						+	+	
16				+				+			+
17					+			+			+
18						+		+			+
19							+	+			+
20						+		+			+
21					+			+		+	
22				+					+	+	
23			+					+		+	
24		+							+	+	
25	+							+		+	
26		+							+		+
27			+					+			+
28				+					+		+
29					+			+			+
30						+			+		+
31							+	+		+	
32						+			+	+	
33					+			+		+	

Требования к реализации:

- приложение должно иметь многофайловую структуру: включать заголовочные файлы и файлы реализации заданных фигур;
- заголовок приложения должен содержать номер группы и вариант задания, например: **22VP1_1**;
- приложение должно обрабатывать исключительные ситуации, связанные с изображением геометрических фигур: некорректные координаты (например, отрицательные значения координат), выход за границы окна отображения и др.;
- параметры классов фигур должны соответствовать заданным параметрам из предметной области разработки (см. рис. 36);
- для каждой фигуры следует задать цвет линии и цвет фона.

4. ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ И ЗАЩИТЫ КУРСОВОГО ПРОЕКТА

4.1. Раздел «Анализ предметной области и функциональных требований»

1. Чем определяется предметная область разработки?
2. Что такое модель предметной области?
3. Чем отличается модель предметной области от диаграммы программных классов?
4. Каково назначение диаграммы вариантов использования?
5. Что такое вариант использования и как он трансформируется в реализуемом приложении?
6. Как контекст системы отображается в диаграмме вариантов использования?
7. В каких случаях целесообразно использовать отношение расширения?
8. В каких случаях целесообразно использовать отношение включения?
9. Что содержит сценарий варианта использования?
10. Какая диаграмма используется для графического отображения сценария?

4.2. Раздел «Проектирование»

11. Чем определяется структура разрабатываемого приложения?
12. Что такое статическая структура программных средств и что ей соответствует в языке UML?
13. Какая разница между определением класса и определением объекта?
14. Какие отношения между классами отображаются на диаграмме программных классов? Приведите примеры из своего курсового проекта.
15. Наследуются ли члены и методы базового класса в последующих поколениях производных классов? Приведите примеры из своего курсового проекта.
16. С какой целью используется служебное слово **protected**?
17. Что такое таблица виртуальных методов? Покажите ее в структуре классов вашего курсового проекта.

18. Всегда ли целесообразно использовать виртуальные методы?
19. Что такое чисто виртуальная функция и в чем ее отличие от просто виртуальной функции?
20. Что «делает» класс абстрактным и может ли такой класс иметь реализацию своих функций?
21. В чем отличие отношений Агрегация и Композиция?

4.3. Раздел «Реализация»

22. Что отображает диаграмма компонентов?
23. Какие стереотипы артефактов вы знаете? Приведите примеры использования этих стереотипов в своем курсовом проекте.
24. В чем заключается многофайловая структура приложения?
25. С какой целью создается заголовочный файл? Приведите пример в своем курсовом проекте.
26. Для чего используется препроцессорный оператор **include**?
27. Что такое пространство имен и для чего используется пространство **std**?
28. Что такое исключение и класс исключения?
29. Может ли класс исключения содержать члены и, если да, для чего?
30. Как создается объект исключения?
31. Для чего используются операторы **try**, **catch**?

4.4. Раздел «Тестирование»

32. С какой целью выполняется тестирование?
33. Можно ли осуществить полное тестирование?
34. Что такое функциональное тестирование и как оно связано с вариантами использования? Приведите примеры в своем курсовом проекте.
35. Что означает выражение «тест пройден»?
36. Сколько тестов необходимо, чтобы проверить выход графического изображения фигуры за границу окна отображения? Приведите примеры этих тестов в своем курсовом проекте.
37. Означает ли правильная обработка исключительной ситуации, что тест пройден? Если да, то почему?

Приложение А
КОД ПРОГРАММЫ
(обязательное)

Главная программа

```
int main() {
    Figure* s[2];
    s[0] = new Rectangle(5, 5, 10, 5);
    s[1] = new Circle(15, 15, 10);
    s[0]->print(); cout << "    area= "<<s[0]->area() << endl;
    s[1]->print(); cout << "    area= "<<s[1]->area() << endl;
    cout << endl << "    vector: " << endl;
    vector<Figure*> v;
    v.push_back(new Rectangle(25, 25, 50, 50));
    v.push_back(new Circle(100, 100, 50));
    for (auto i : v) {
        i->print(); cout << endl;
    }
    cout << endl << "    list: " << endl;
    list<Figure*> ls;
    ls.push_back(new Rectangle(25, 25, 50, 50));
    ls.push_back(new Circle(100, 100, 50));
    for (auto i : ls) {
        i->print(); cout << endl;
    }
    cout << endl << "    set: " << endl;
    set<Figure*> st;
    st.insert(new Rectangle(25, 25, 50, 50));
    st.insert(new Circle(100, 100, 50));
    for (auto i : st) {
        i->print(); cout << endl;
    }
    cout << endl << "    stack_vector: " << endl;
    stack<Figure*, vector<Figure*> > sv;
    sv.push(new Rectangle(25, 25, 50, 50));
    sv.push(new Circle(100, 100, 50));
    while(!sv.empty()) {
        sv.top()->print(); cout << endl; sv.pop();
    }
    cout << endl << "    queue_list: " << endl;
    queue<Figure*, list<Figure*> > q;
    q.push(new Rectangle(25, 25, 50, 50));
    q.push(new Circle(100, 100, 50));
    while (!q.empty()) {
        q.front()->print(); cout << endl; q.pop();
    }
    system("pause");
    return 0;
}
```

Приложение Б
ГРАФИЧЕСКАЯ ЧАСТЬ
(обязательное)

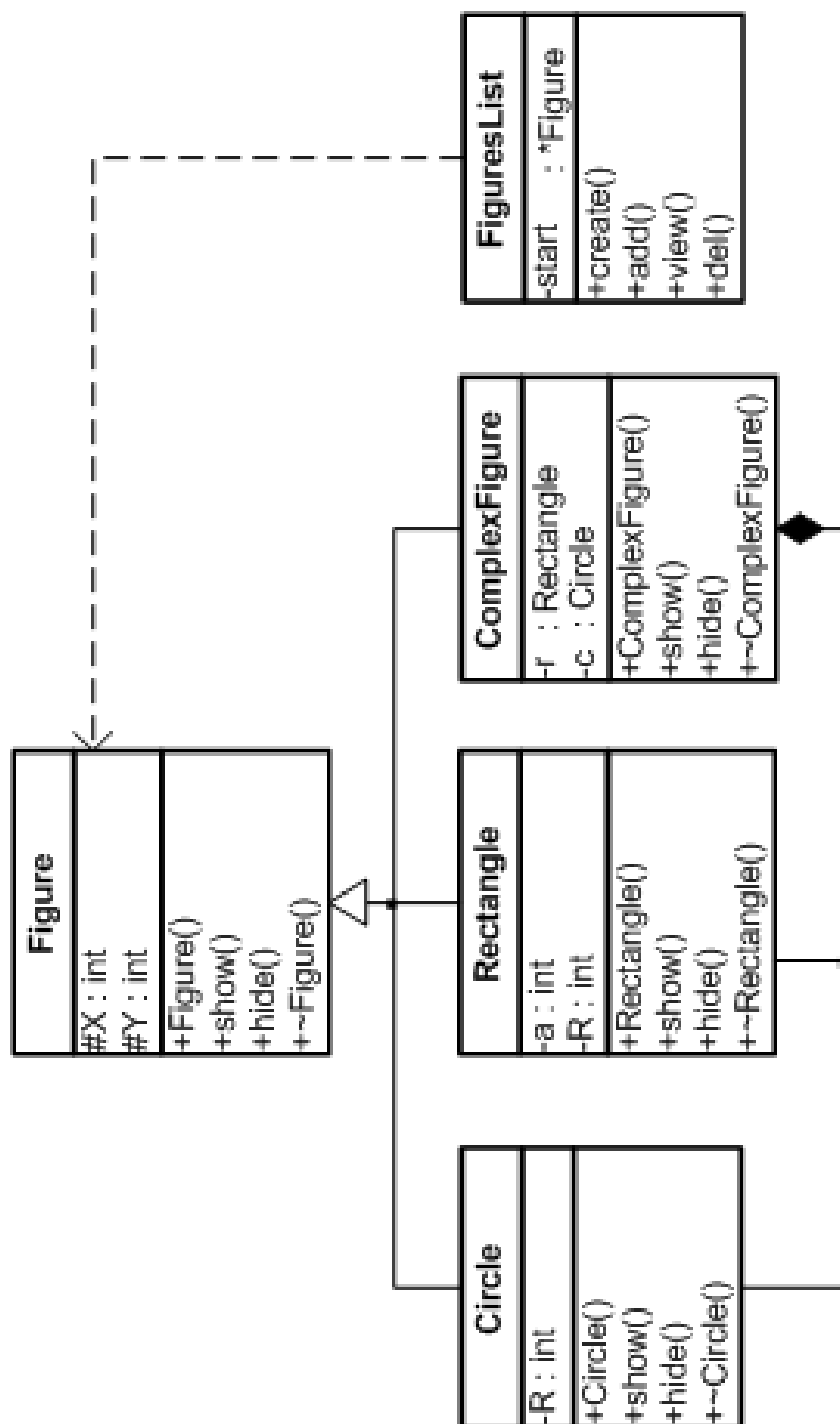


Рис. Б1. Диаграмма классов

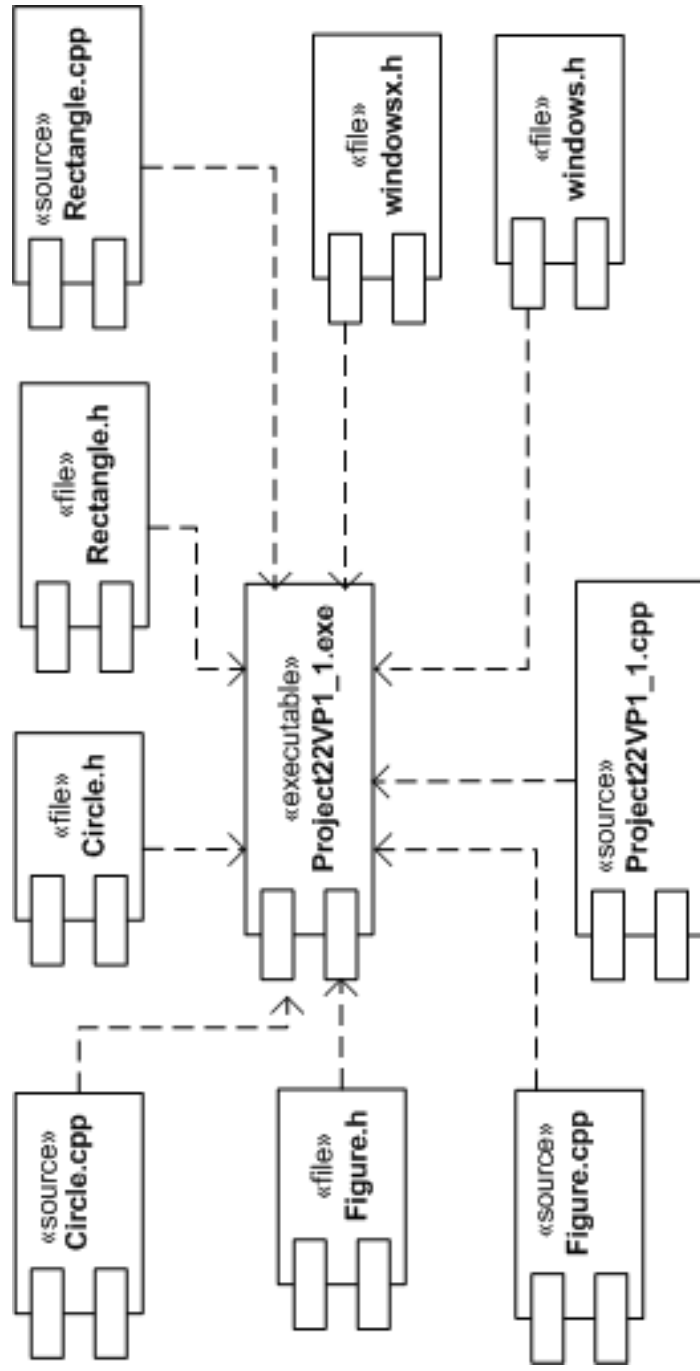


Рис. Б2. Диаграмма компонентов

Учебное издание

**ПРОГРАММИРОВАНИЕ
НА ЯЗЫКЕ C++.
КУРСОВОЕ ПРОЕКТИРОВАНИЕ**

Гурьянов Лев Вячеславович,
Гурьянова Людмила Станиславовна,
Дзюба Елена Анатольевна,
Самуйлов Сергей Владимирович,
Афонин Александр Юрьевич

Редактор *А. Г. Темникова*
Технический редактор *Р. Б. Бердникова*
Компьютерная верстка *Р. Б. Бердниковой*
Дизайн обложки *И. В. Шваревой*

Подписано в печать 15.06.2023. Формат 60×84¹/₁₆.
Усл. печ. л. 4,42.
Заказ № 264. Тираж 68.

Издательство ПГУ.
440026, г. Пенза, ул. Красная, 40.
Тел.: (8412) 66-60-49, 66-67-77; e-mail: iic@pnzgu.ru