МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тульский государственный университет»

Институт прикладной математики и компьютерных наук

Кафедра вычислительной техники

Курсовая работа по дисциплине

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

на тему: Исследование вычислительной эффективности объектно-ориентированных приложений.

Студент группы 220681 Хохряков Д.А. \_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.) (Подпись, дата)

Руководитель работы к.т.н., доц. Берсенев Г.Б. \_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О., должность) (Подпись, дата)

Комиссия: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тула, 2020

МИНОБРНАУКИ РОССИИ

ФГБОУ ВО «Тульский государственный университет»

Институт прикладной математики и компьютерных наук

Кафедра «Вычислительная техника»

**З А Д А Н И Е**

На курсовую работу по дисциплине «Объектно-ориентированное программирование»

студенту группы 220681 Хохрякову Даниилу Андреевичу

Тема работы:

«Исследование вычислительной эффективности объектно- ориентированных приложений»

Входные данные Вариант №17:

задача: определение площади методом Монте-Карло;

фигура: debo;

приложение №1: процедурное приложение на базе Console/C#;

приложение №2: объектно-ориентированное приложение на базе Console/C#.

Задание получил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «22» февраля 2020.

(подпись студента)

Срок предоставления задания «01» июня 2020.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка подписи)

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г.

К защите. Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка подписи)

Замечания руководителя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г.

*При защите курсового проекта (работы) наличие рецензии обязательно.*

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc42529201)

[1. ПОСТАНОВКА ЗАДАЧИ 6](#_Toc42529202)

[2. РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ 8](#_Toc42529203)

[2.1. Анализ вариантов использования программы 8](#_Toc42529204)

[2.2. Требования к приложению 9](#_Toc42529205)

[3. АНАЛИЗ АЛГОРИТМА РЕШЕНИЯ 10](#_Toc42529206)

[3.1. Метод Монте-Карло 10](#_Toc42529207)

[3.2. Определение попадания точки внутрь фигуры 10](#_Toc42529208)

[4. ПРОЦЕДУРНОЕ ПРИЛОЖЕНИЕ НА БАЗЕ Console/C# 12](#_Toc42529209)

[4.1. Программные средства разработки 12](#_Toc42529210)

[4.2. Логическое проектирование 12](#_Toc42529211)

[4.3. Системные требования 12](#_Toc42529212)

[4.4. Структура приложения 12](#_Toc42529213)

[4.5. Результаты работы 14](#_Toc42529214)

[5. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРИЛОЖЕНИЕ НА БАЗЕ Console/C# 17](#_Toc42529215)

[5.1. Программные средства разработки 17](#_Toc42529216)

[5.2. Логическое проектирование 17](#_Toc42529217)

[5.3. Описание программы 18](#_Toc42529218)

[5.4. Руководство системного программиста 20](#_Toc42529219)

[5.5. Руководство программиста 24](#_Toc42529220)

[5.6. Руководство оператора 25](#_Toc42529221)

[5.7. Результаты работы 26](#_Toc42529222)

[6. АНАЛИЗ ВЫЧИСЛИТЕЛЬНОЙ ЭФФЕКТИВНОСТИ ПРИЛОЖЕНИЙ 28](#_Toc42529223)

[7. УЛУЧШЕНИЕ ХАРАКТЕРИСТИК КАЧЕСТВА ПРИЛОЖЕНИЙ 29](#_Toc42529224)

[7.1. Использование встроенной оптимизации кода 29](#_Toc42529225)

[7.2. Профилирование 30](#_Toc42529226)

[7.3. Результаты улучшения характеристик 31](#_Toc42529227)

[ЗАКЛЮЧЕНИЕ 32](#_Toc42529228)

[СПИСОК ЛИТЕРАТУРЫ 33](#_Toc42529229)

[ПРИЛОЖЕНИЕ 34](#_Toc42529230)

[Файл Program.cs (процедурное приложение)](#_Toc42529231)

[Файл Point.cs](#_Toc42529232)

[Файл Program.cs (объектно-ориентированное приложение)](#_Toc42529233)

[Файл Point.cs](#_Toc42529234)

[Файл Trinagle.cs](#_Toc42529235)

# ВВЕДЕНИЕ

В настоящее время объектно-ориентированное программирование (ООП) является доминирующим стилем при создании больших программ и программных систем. Процедурно-ориентированное программирование, широко использовавшееся до появления ООП, обычно позволяет создавать более эффективные в вычислительном отношении реализации приложений, что является существенным фактором при разработке систем реального времени. На практике эти два стиля программирования часто используются совместно, позволяя варьировать степень их применения в программах.

Использование объектно-ориентированного (ОО) подхода при разработке программного обеспечения позволяет преодолеть естественную сложность разрабатываемого ПО, упростить процесс отладки и последующего сопровождения, расширения и переноса ПО на другие платформы.

ОО подход включает в себя объектно-ориентированный анализ (ООА), дизайн (проектирование) (ООД) и программирование.

Объектно-ориентированный анализ – это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

Объектно-ориентированное проектирование – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, статической и динамической моделей проектируемой системы.

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Идеальное ОО приложение должно быть расширяемым, масштабируемым, сопровождаемым и переносимым.

Основным понятием ООП является класс. Класс (class) определяет группу объектов с общими свойствами (атрибутами), поведением (функциями), семантикой и связями с другими объектами. Класс можно трактовать как шаблон для создания объектов. Каждый объект является экземпляром некоторого класса, причем только одного. Класс может наследовать один или нескольких интерфейсов, реализуя свойства, события и методы каждого из них.

Основными концепциями ООП, которыми руководствуются при создании классов, являются инкапсуляция, наследование и полиморфизм (параметрический и основной, применяемый при наследовании). При создании сложных объектов наряду с наследованием (отношением «is-a») широко используется включение объектов (отношение «is-part-of»)).

Существует огромное количество методологий и рекомендаций, направленных на повышение эффективности процесса проектирования программных систем. Среди них можно выделить принципы SOLID для гибкого проектирования объектно-ориентированного ПО.

C# – объектно-ориентированный язык программирования. Он относится к семье языков с C-подобным синтаксисом. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

В данной курсовой работе создаются процедурно-ориентированная и объектно-ориентированная реализации конкретного приложения и проводится сравнительный анализ их вычислительной эффективности.

# ПОСТАНОВКА ЗАДАЧИ

В результате выполнения курсовой работы необходимо создать и сравнить по вычислительной эффективности два приложения, решающие задачу приближенного вычисления площади геометрической фигуры методом Монте-Карло с использованием процедурного и объектно-ориентированного программирования соответственно. Также нужно выполнить исследование вычислительной эффективности созданных приложений и представить результаты в виде таблиц, провести анализ и улучшить заданные характеристики качества приложений.

Вариант задания №17:

* геометрическая фигура: ;
* приложение 1: процедурное приложение на базе Console/C#;
* приложение 2: объектно-ориентированное приложение на базе Console/C#;

Геометрическая фигура представляет собой объединение треугольника и сектора с центром (рис. 1).

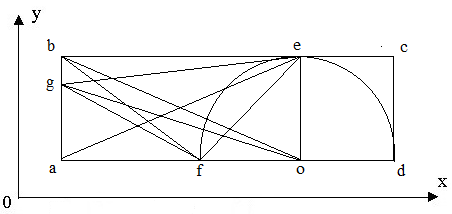


Рис. 1. Фигура

Исходными данными для каждого приложения являются координаты вершин треугольника. Для однозначного определения расположения фигуры необходимо задать две точки: и . Для определения площади фигуры необходимо вначале вычислить площадь прямоугольника, описывающего заданную фигуру, а затем раз сгенерировать по два случайных числа для координат и , определяющие точку внутри прямоугольника. Генерируемые случайным образом точки должны равномерно заполнять площадь прямоугольника. Для этого случайные числа должны иметь равномерное распределение (по ширине и высоте прямоугольника соответственно).

Для каждой точки выполняется проверка, попала ли точка внутрь заданной фигуры. Если из точек точек оказалось внутри фигуры, а площадь прямоугольника равна , то площадь фигуры будет приближенно равна .

Задачами курсовой работы являются:

* приобретение навыков решения вычислительных задач;
* практическое освоение современных инструментальных систем разработки ПО;
* сравнительный анализ вычислительной эффективности процедурных и объектно-ориентированных программ;
* получение навыков создания программ на языке C# платформы Microsoft .NET Framework;
* приобретение практических навыков оформления и выпуска документации в соответствии с требованиями стандартов (ЕСПД, UML).

# РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ

## 2.1. Анализ вариантов использования программы

Необходимо разработать программу, которая вычисляет площадь фигуры методом Монте-Карло. Программа должна предоставить пользователю возможность выполнения следующих действий:

* ввод координат (вручную или из контрольного примера);
* нахождение площади фигуры методом Монте-Карло;
* вывод результатов на экран;
* ввод новых значений;
* повторный расчёт;
* выход.

Диаграмма вариантов использования приложений представлена на рис. 2. В диаграмме были использованы такие виды отношений как расширения (пунктирная стрелка с подписью расширить), включение (пунктирная стрелка с подписью включить) и наследование (стрелка с не закрашенным треугольником). Расширение означает, что процедура может выполняется в зависимости от некоторых условий. Включение указывает на то, что одна процедура включается в некоторой точке в другую в качестве составного компонента. Наследование представляет собой связь между родителем и потомком, который наследует все его свойства.



Рис. 2. Диаграмма вариантов использования приложений

## 2.2. Требования к приложению

**2.2.1. Общие требования**

* Программа должна поддерживать выполнение следующих команд: ввод, расчёт, выход.
* Вводимые координаты представляют собой пару значений *x* и *y* для каждой точки фигуры. Данные вводятся пользователем через консоль и представляют собой значения типа double.
* При вводе значений необходимо проверять корректность вводимых данных. Также нужно проверять, соответствуют ли вершины форме фигуры .
* После нахождения площади треугольника приложение должно выводить в консоль количество точек , количество попавших точек, найденную площадь, погрешность и время работы для .

**2.2.2. Требования к процедурному приложению**

* Фигура должна описываться с помощью точек, представленных в виде локальных переменных.
* Все методы приложения должны располагаться в одном статическом классе.

**2.2.3. Требования к объектно-ориентированному приложению**

* Фигура должна описываться с помощью предназначенных для этого классов (Triangle, Circle, Point).
* Каждый класс должен иметь три типа конструкторов: по умолчанию, инициализации, копии.

# АНАЛИЗ АЛГОРИТМА РЕШЕНИЯ

Пусть координаты точек заданной фигуры (рис. 3) объявлены следующим образом: *, , ,* .

Координаты всех точек задаются пользователем.

## 3.1. Метод Монте-Карло

Впишем исходную фигуру в прямоугольник , площадь которого:

.

Так как , , , а , то площадь прямоугольника можно записать в виде:

.

Сгенерируем в этом прямоугольнике точек, координаты которых выбираются случайным образом.

Определим число точек , которые попали в фигуру.

Приближённую площадь фигуры вычислим по следующей формуле:

*.*

Для нахождения точной площади фигуры используется следующая формула:

,

где .

Для нахождения погрешности вычислений в процентах методом Монте-Карло можно воспользоваться следующей формулой:

.

## 3.2. Определение попадания точки внутрь фигуры

Необходимо сгенерировать внутри прямоугольника точку и определить, принадлежит ли эта точка фигуре , если известны координаты точек фигуры. Пример расположения точек фигуры и точки представлен на рис. 3.

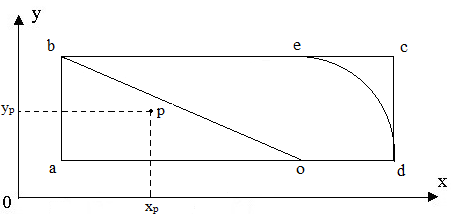


Рис. 3. Точки, необходимые для расчётов

Условие нахождения точки между векторами и :

, , , , , .

Уравнение окружности:

.

Квадрат расстояния от точки до точки :

.

Для определения попадания точки внутрь фигуры используется следующий алгоритм:

1. если , то точка принадлежит фигуре, если ;
2. если , то точка принадлежит фигуре, если ;
3. если , то точка принадлежит фигуре, если и.

# ПРОЦЕДУРНОЕ ПРИЛОЖЕНИЕ НА БАЗЕ Console/C#

## 4.1. Программные средства разработки

Процедурное приложение выполняется в инструментальной системе MS Visual Studio 19. Она предоставляет системную объектно-ориентированную среду на базе платформы Microsoft .NET Framework для разработки настольных (клиентских) приложений. Для разработки приложения используется предоставленный платформой класс System.Console для создания консольных приложений.

## 4.2. Логическое проектирование

Все методы, которые используются для нахождения площади, расположены в основном классе Program, в котором также располагается основной метод Main. Точки, создаваемые в методах, являются объектами класса Point и содержат поля, описывающие координаты и .

## 4.3. Системные требования

Для корректной работы приложения требуется:

* Операционная система Windows 7 и выше;
* платформа .NET Framework 4.5.2 и выше;
* архитектура 32-разрядная (х86)/64-разрядная (х64);
* оперативная память 512 МБ и более;
* устройства ввода: клавиатура и мышь.

## 4.4. Структура приложения

Диаграмма класса Program представлена на рис. 4.



Рис. 4. Диаграмма класса Program

Приложение одержит следующие функции:

public static void Main() – основной метод, вызывается в самом начале и в конце каждого этапа для обработки команд пользователя;

private static void Manual() – получает координаты двух точек фигуры из консоли и попутно проверяет корректность введённых данных. Если данные введены некорректно, приложение попросит ввести координаты ещё раз. После находит с помощью метода MonteCarlo площадь и выводит результаты на консоль;

private static void Test() – получает контрольные координаты точек фигуры. После находит с помощью метода MonteCarlo площадь и выводит результаты на консоль;

private static void MonteCarlo(Point d, Point e, Point b, Point o) – вычисление площади методом Монте-Карло;

private static bool IsPointInTriangle(Point p, Point b, Point e, Point o) – проверяет, находится ли точка внутри треугольника ;

private static bool IsPointInCircle(Point p, Point o, double R) – проверяет, находится ли точка внутри круга с центром в точке радиуса ;

Более детально алгоритмы методов MonteCarlo, IsPointInCircle и IsPointInTriangle можно рассмотреть на рис. 5-7.

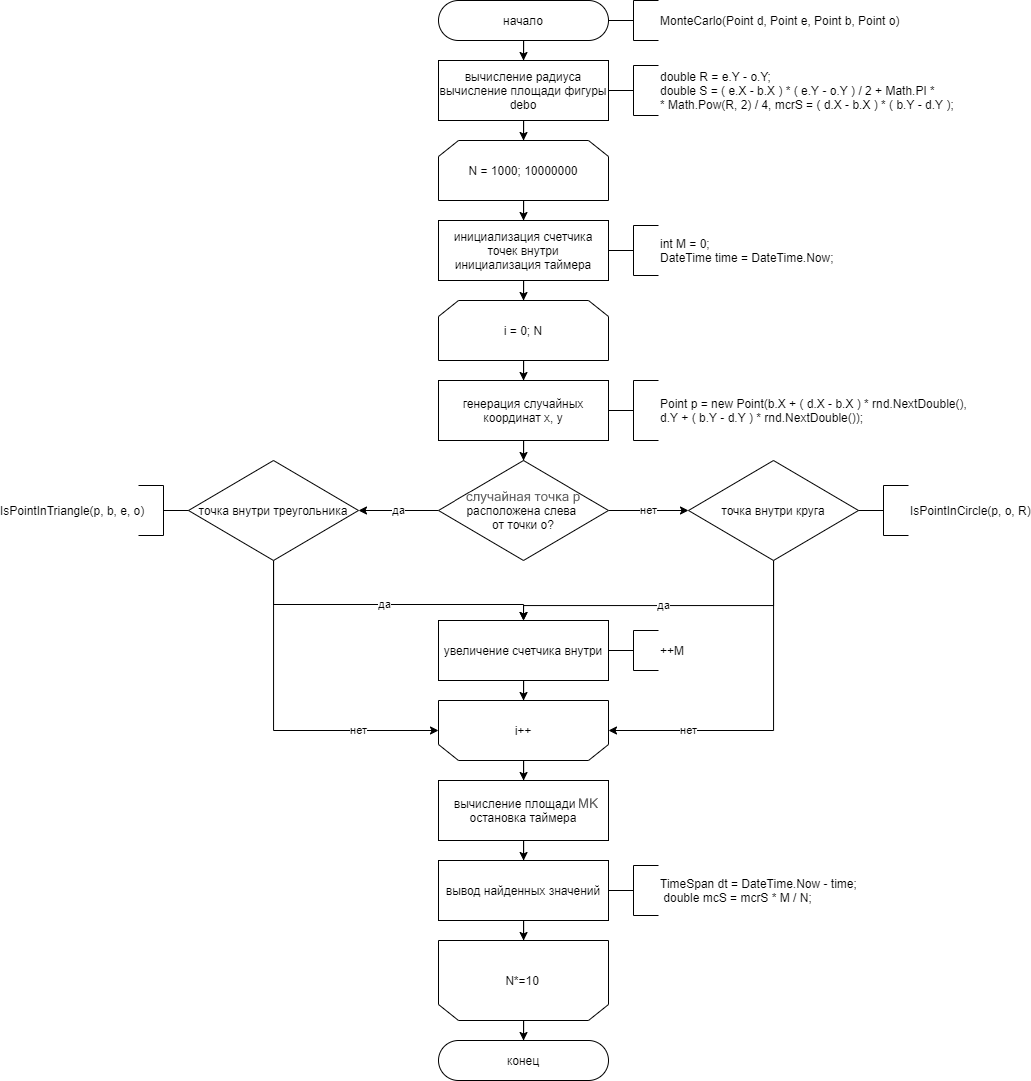


Рис. 5. Схема метода MonteCarlo класса Program

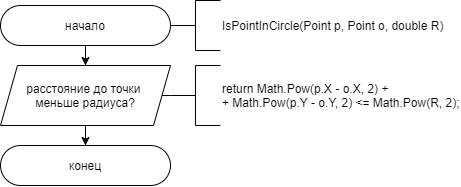


Рис. 6. Схема метода IsPointInCircle класса Program

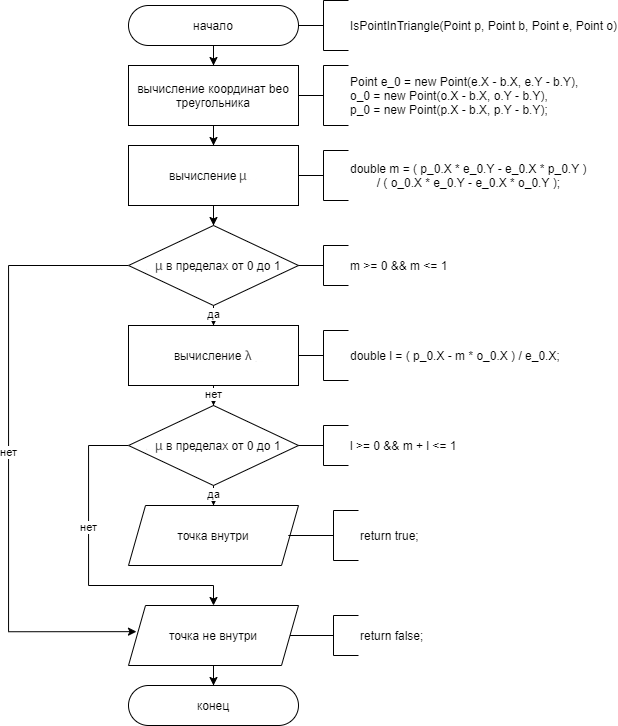


Рис. 7. Схема метода IsPointInTriangle класса Program

## 4.5. Результаты работы

Результаты работы приложения представлены на рис. 8.

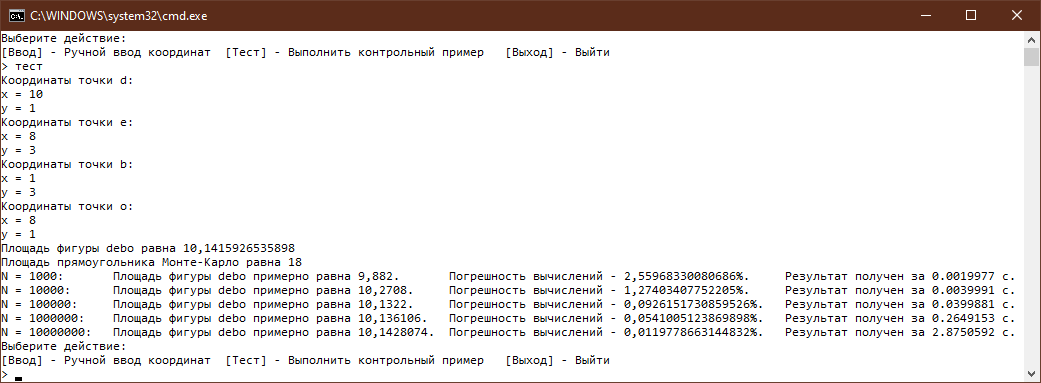


Рис. 8. Тестирование процедурного приложения

# ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРИЛОЖЕНИЕ НА БАЗЕ Console/C#

## 5.1. Программные средства разработки

Объектно-ориентированное приложение выполняется в инструментальной системе MS Visual Studio 19. Она предоставляет системную объектно-ориентированную среду на базе платформы Microsoft .NET Framework для разработки настольных (клиентских) приложений. Для разработки приложения используется предоставленный платформой класс System.Console для создания консольных приложений.

## 5.2. Логическое проектирование

В рассматриваемой предметной области можно выделить следующие сущности: точка, прямоугольный треугольник, круг, прямоугольник, фигура. Размеры и форма треугольника задаются тремя точками. Каждая точка задаётся двумя координатами: и . Для сущности точки можно создать класс Point, который будет содержать конструкторы и поля , . Для сущности треугольник можно создать класс Triangle, который будет содержать конструкторы и три объекта класса Point. Для сущности круг можно создать класс Circle, который будет наследоваться от класса Point и содержать конструкторы и объект типа double. Для сущности прямоугольник можно создать класс Rect, который будет наследоваться от класса Point и содержать конструкторы и два объекта типа double. Для сущности фигура можно создать класс Figure, который будет содержать конструкторы и объекты классов Circle, Triangle и Rectangle. Для удобства метод, участвующий в нахождении площади методом Монте-Карло, размещается в классе Figure. Диаграмма классов объектно-ориентированного приложения представлена на рис. 9.

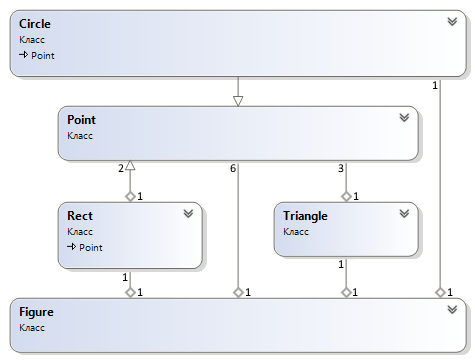


Рис. 9. Диаграмма классов объектно-ориентированного приложения

## 5.3. Описание программы

Класс Point представляет собой точку с двумя координатами. Диаграмма класса представлена на рис. 10.



Рис. 10. Диаграмма класса Point

Класс Point содержит следующие свойства:

public double X {get; private set;} – координата *x* точки;

public double Y {get; private set;} – координата *y* точки.

Класс Point содержит следующие конструкторы:

public Point() – по умолчанию;

public Point(double x, double y) – инициализации;

public Point(Point p) – копии.

Класс Circle, наследуемый от класса Point, представляет собой круг с центром и радиусом. Диаграмма класса представлена на рис. 11.



Рис. 11. Диаграмма класса Circle

Класс Circle содержит следующие свойства:

double R – радиус круга.

Класс Circle содержит следующие конструкторы:

public Circle() – по умолчанию;

public Circle(Point p, double R) – инициализации;

public Circle(Circle c) – копии.

Класс Circle содержит следующие методы:

private bool IsPointHere(Point p) – проверяет, находится ли точка внутри этого круга;

private double Square() – находит точную площадь круга.

Класс Triangle представляет собой три точки с методами по поиску площади. Диаграмма класса представлена на рис. 12.



Рис. 12. Диаграмма класса Triangle

Класс Triangle содержит следующие свойства:

Point B – вершина ;

Point E – вершина ;

Point O – вершина .

Класс Triangle содержит следующие конструкторы:

public Triangle() – по умолчанию;

public Triangle(Point b, Point e, Point o) – инициализации;

public Triangle(Triangle t) – копии.

Класс Triangle содержит следующие методы:

private bool IsPointHere(Point p) – проверяет, находится ли точка внутри этого треугольника;

private double Square() – находит точную площадь прямоугольного треугольника по двум катетам.

Класс Rect, наследуемый от класса Point, представляет собой прямоугольник с вершиной, шириной и высотой. Диаграмма класса представлена на рис. 13.



Рис. 13. Диаграмма класса Rect

Класс Rect содержит следующие свойства:

double DX – ширина прямоугольника;

double DY – высота прямоугольника.

Класс Rect содержит следующие конструкторы:

public Rect() – по умолчанию;

public Rect(Point p0, Point p) – инициализации;

public Rect(Rect r) – копии.

Класс Rect содержит следующие методы:

private double Square() – находит точную площадь прямоугольника.

Класс Figure, представляет собой фигуру, состоящую из треугольника, сектора и описанного вокруг них прямоугольника. Диаграмма класса представлена на рис. 14.



Рис. 14. Диаграмма класса Rect

Класс Figure содержит следующие свойства:

Circle c – круг;

Triangle t – треугольник;

Rect r – прямоугольник.

Класс Figure содержит следующие конструкторы:

public Figure() – по умолчанию;

public Figure(Point d, Point e, Point b, Point o) – инициализации;

public Figure(Figure f) – копии.

Класс Figure содержит следующие методы:

private bool IsPointHere(Point p) – проверяет, находится ли точка внутри этой фигуры;

private double Square() – находит точную площадь фигуры;

private double MonteCarlo() – находит площадь фигуры методом Монте-Карло.

Диаграмма класса Program представлена на рис. 15.



Рис. 15. Диаграмма класса Program

Приложение содержит следующие функции:

public static void Main() – основной метод, вызывается в самом начале и в конце каждого этапа для обработки команд пользователя;

private static void Manual() – получает координаты двух точек фигуры из консоли и попутно проверяет корректность введённых данных. Если данные введены некорректно, приложение попросит ввести координаты ещё раз. После находит с помощью метода MonteCarlo площадь и выводит результаты на консоль;

private static void Test() – получает контрольные координаты точек фигуры. После находит с помощью метода MonteCarlo площадь и выводит результаты на консоль.

## 5.4. Руководство системного программиста

**5.4.1. Общие сведения о приложении**

Приложение разработано для вычисления площади фигуры методом Монте-Карло. Данное приложение реализовано как консольное на языке программирования C# с использованием объектно-ориентированного подхода.

**5.4.2. Системные требования**

Для корректной работы приложения требуется:

* Операционная система Windows 7 и выше;
* платформа .NET Framework 4.5.2 и выше;
* архитектура 32-разрядная (х86)/64-разрядная (х64);
* оперативная память 512 МБ и более;

устройства ввода: клавиатура и мышь.

**5.4.3. Структура программы**

Программа состоит из шести классов. Класс Program содержит главный метод Main, который вызывается в самом начале и в конце каждого этапа для обработки команд пользователя, методы Test и Manual, создающие объекты класса Point. Класс Figure содержит метод MonteCarlo, выполняющий поиск площади методом Монте-Карло. Класс Point содержит координаты точки *x*, *y* и конструкторы. Класс Circle, наследуемый от Point, содержит один объект типа double и методы, которые отвечают за создание класса и поиск площади методом Монте-Карло. Класс Triangle содержит три объекта класса Point и методы, которые отвечают за создание класса и поиск площади методом Монте-Карло. Класс Rect, наследуемый от Point, содержит два объекта типа double и методы, которые отвечают за создание класса.

**5.4.4. Проверка программы**

Для тестирования программы можно выбрать координаты фигуры по умолчанию. В результатах вычисления программы площадь фигуры должна составлять единиц.

**5.4.5. Сообщения системному программисту**

В ходе выполнения программы возможен вывод следующего сообщения об ошибке:

* Ошибка! Недопустимое расположение точек и !

При появлении этой ошибки необходимо повторно ввести координаты точки и , так как введённые координаты не соответствуют фигуре . Чтобы фигура соответствовала , необходимо выполнение следующих условий: *, ,* .

## 5.5. Руководство программиста

**5.5.1. Назначение и условие применения программы**

Приложение разработано для вычисления площади фигуры методом Монте-Карло. Программа поддерживает выполнение следующих функций:

* ввод координат (вручную или из контрольного примера);
* нахождение площади фигуры методом Монте-Карло и вывод результатов на экран;
* ввод новых значений;
* повторный расчёт;
* выход.

Для корректной работы приложения требуется:

* Операционная система Windows 7 и выше;
* платформа .NET Framework 4.5.2 и выше;
* архитектура 32-разрядная (х86)/64-разрядная (х64);
* оперативная память 512 МБ и более;
* устройства ввода: клавиатура и мышь.

**5.5.2 Характеристики** **программы**

За вычетом времени, затрачиваемого пользователем на ввод координат и выбор команд, программа выполняется за 3.5 секунды.

**5.5.3 Входные и выходные данные**

Вся входная информация (координаты вершин и команды) получается программой из консоли. Вся выходная информация (сообщения и результаты) выводится программой в консоль.

**5.5.4 Сообщения**

В ходе выполнения программы возможен вывод следующего сообщения об ошибке:

* Ошибка! Недопустимое расположение точек и !

При появлении этой ошибки необходимо повторно ввести координаты точки и , так как введённые координаты не соответствуют фигуре . Чтобы фигура соответствовала , необходимо выполнение следующих условий: *, ,* .

## 5.6. Руководство оператора

**5.6.1 Назначение программы**

Приложение разработано для вычисления площади фигуры методом Монте-Карло. Программа поддерживает выполнение следующих функций:

* ввод координат (вручную или из контрольного примера);
* нахождение площади фигуры методом Монте-Карло и вывод результатов на экран;
* ввод новых значений;
* повторный расчёт;
* выход.

**5.6.2. Условия выполнения программы**

Для корректной работы приложения требуется:

* Операционная система Windows 7 и выше;
* платформа .NET Framework 4.5.2 и выше;
* архитектура 32-разрядная (х86)/64-разрядная (х64);
* оперативная память 512 МБ и более;

устройства ввода: клавиатура и мышь.

**5.6.3. Выполнение программы**

В самом начале и в конце каждого этапа работы программа предлагает пользователю ввести три команды: ввод, тест, выход.

При вводе команды «ввод» пользователю необходимо ввести координаты и для каждой точки фигуры, после чего программа выполнит поиск площади методом Монте-Карло с разным количеством точек несколько раз и выведет результаты в консоль.

При вводе команды «тест» программа выполнит поиск площади методом Монте-Карло с разным количеством точек несколько раз и выведет результаты в консоль. Вершинам будут заданы координаты по умолчанию.

При вводе команды «выход» приложение будет закрыто.

**5.6.4. Сообщения оператору**

В ходе выполнения программы возможен вывод следующего сообщения об ошибке:

* Ошибка! Недопустимое расположение точек и !

При появлении этой ошибки необходимо повторно ввести координаты точки и , так как введённые координаты не соответствуют фигуре . Чтобы фигура соответствовала , необходимо выполнение следующих условий: *, ,* .

## 5.7. Результаты работы

Результаты работы приложения представлены на рис. 16.

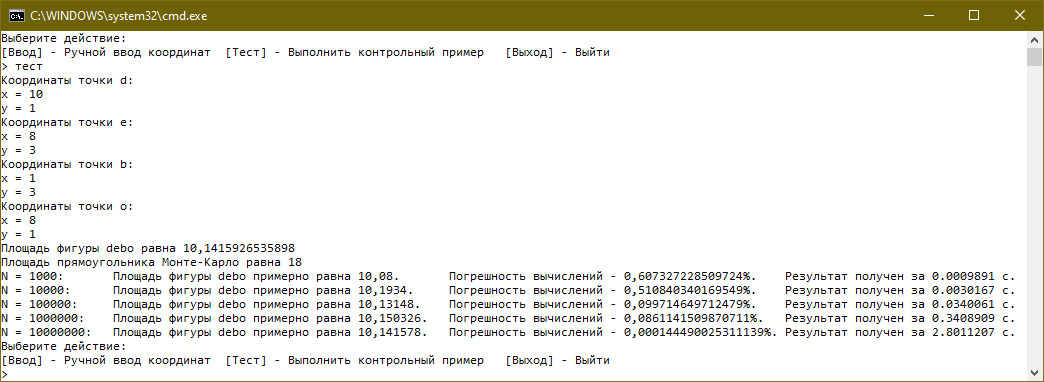


Рис. 16. Тестирование объектно-ориентированного приложения

# АНАЛИЗ ВЫЧИСЛИТЕЛЬНОЙ ЭФФЕКТИВНОСТИ ПРИЛОЖЕНИЙ

Приложения тестировались на компьютере со следующими характеристиками:

* Операционная система Windows 10;
* Процессор Intel(R) Core(TM) i7-4510U с частотой 2.0 GHz;
* Оперативная память 4 ГБ;
* Видеокарта NVIDIA GeForce GT 820M.

Результаты сравнения двух приложений представлены в таблице 1.

Таблица 1 – продолжительность вычислений в двух приложениях

|  |  |  |
| --- | --- | --- |
| Количество точек | Время выполнения, мс | |
| Процедурное приложение на базе Console/C# | Объектно-ориентированное приложение на базе Console/C# |
|  | 0,9987 | 0,9956 |
|  | 2,0326 | 1,9959 |
|  | 8,9769 | 8,9761 |
|  | 71,8081 | 75,8345 |
|  | 710,0631 | 890,5821 |

В среднем оба предложения обладают одинаковой скоростью работы.

# УЛУЧШЕНИЕ ХАРАКТЕРИСТИК КАЧЕСТВА ПРИЛОЖЕНИЙ

## 7.1. Использование встроенной оптимизации кода

После завершения написания приложения необходимо его оптимизировать. Для этого сначала нужно в свойствах проекта (рис. 17) поменять конфигурацию с Debug на Release. Это увеличит производительность приложения из-за того, что в режиме Debug отключены все настройки по оптимизации и часть ресурсов уходит на сбор и хранение информации об отладке.

Также в свойствах проекта во вкладке «Сборка» необходимо отметить пункт «Оптимизировать код» (рис. 17), что сильно скажется на производительности программы.

В этом же окне нужно снять галочку с пункта «Предпочтительна 32‑разрядная версия» (рис. 17). Это серьёзно увеличит производительность, поскольку приложение тестируется на 64-разрядной системе.

В приложении никак не используется класс Trace, поэтому можно снять галочку с пункта «Определить константу TRACE» (рис. 17), что в итоге незначительно увеличит производительность.

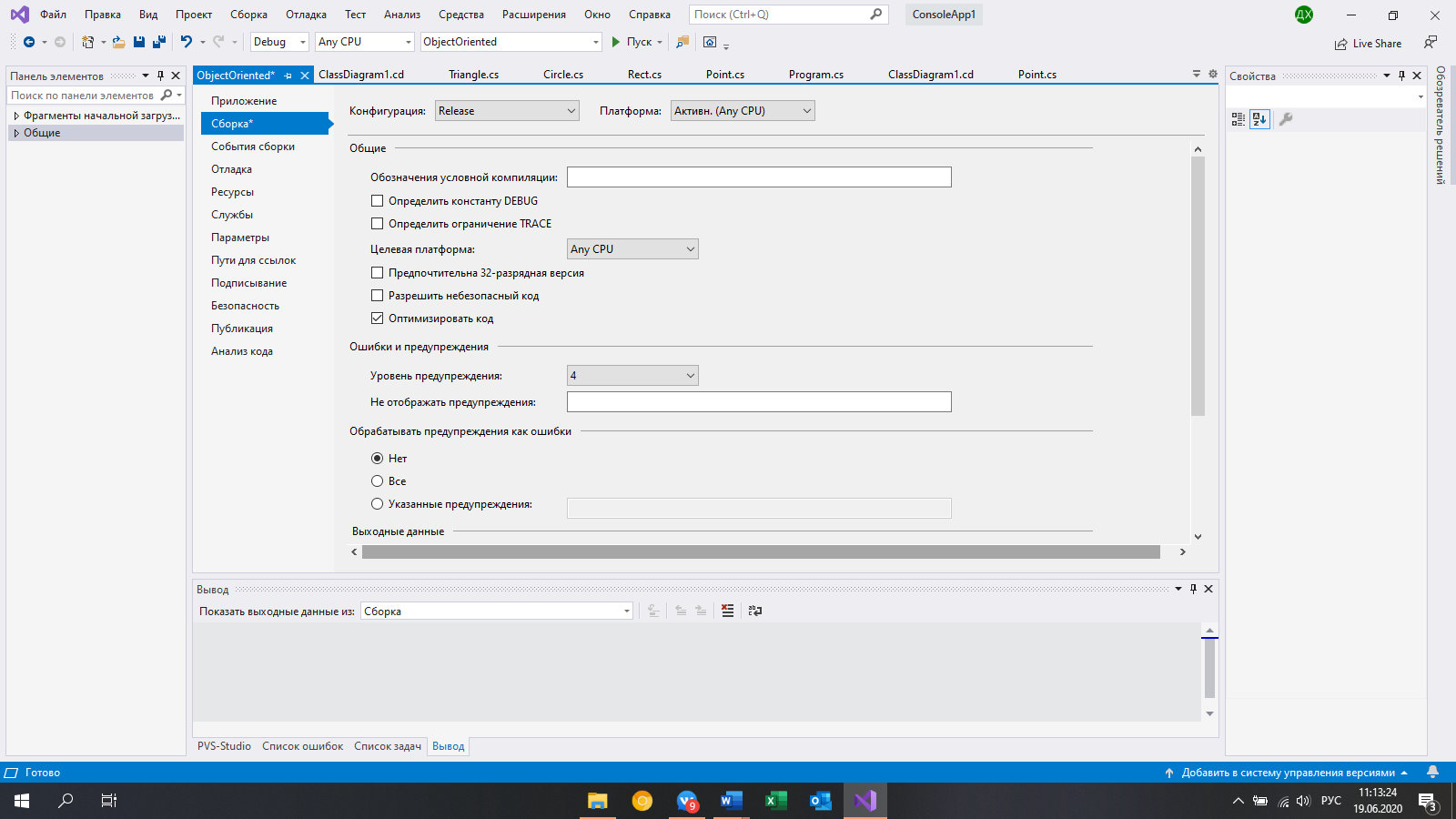


Рис. 17. Вкладка «Сборка» в свойствах проекта

## 7.2. Профилирование

Для приложений было проведено профилирование по использованию ЦП. Результаты приведены на рис. 18 и 19.

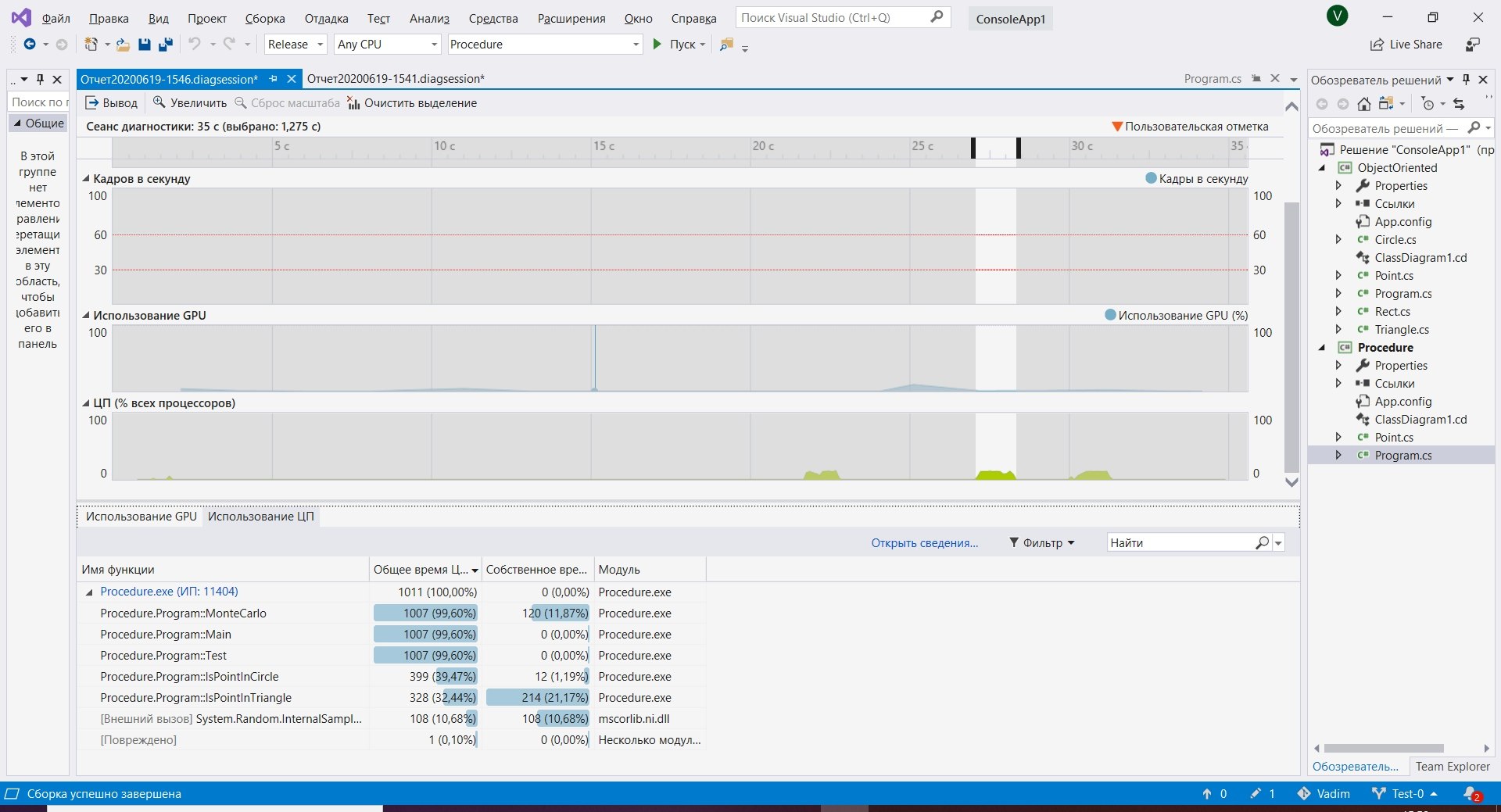


Рис. 18. Результаты профилирования процедурного приложения

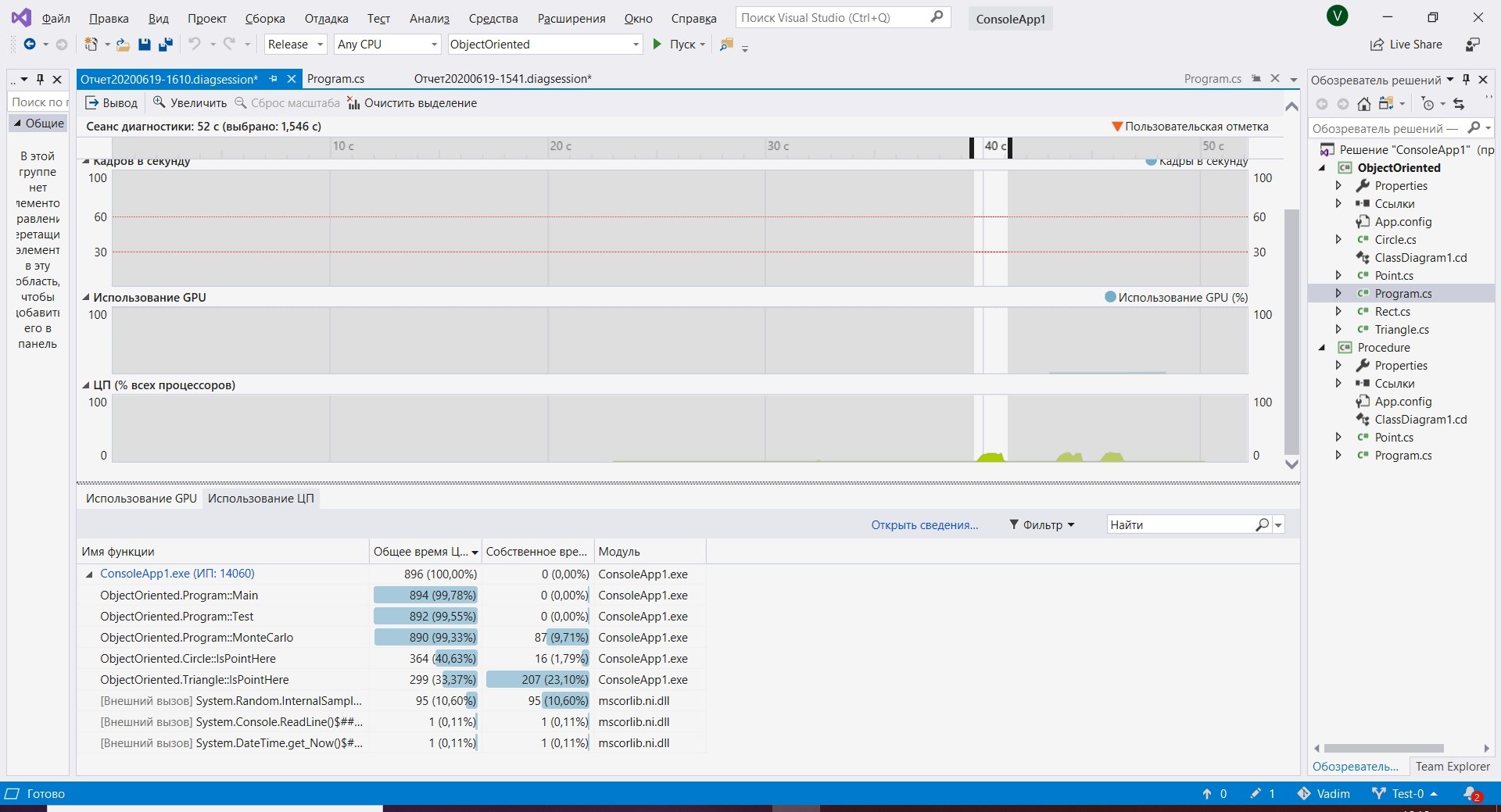


Рис. 19. Результаты профилирования объектно-ориентированного приложения

Из результатов видно, что больше всего ресурсов уходит на метод monteCarloSquare. Также оба приложения примерно одинаково нагружают систему.

## 7.3. Результаты улучшения характеристик

После оптимизации кода производительность приложений сильно возросла. Сравнение результатов до и после оптимизации представлено в таблице 2.

Таблица 2 – сравнение приложений до и после оптимизации

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество точек | Время выполнения, мс | | | |
| Процедурное приложение на базе Console/C# | | Объектно-ориентированное приложение на базе Console/C# | |
| До оптим. | После оптим. | До оптим. | После оптим. |
|  | 1,9977 | 0,9987 | 0,9891 | 0,9956 |
|  | 3,9991 | 2,0326 | 3,0167 | 1,9959 |
|  | 39,9881 | 8,9769 | 34,0061 | 8,9761 |
|  | 264,9153 | 71,8081 | 340,8909 | 75,8345 |
|  | 2875,0592 | 710,0631 | 2801,1207 | 890,5821 |

В среднем процедурное приложение стало работать на 64% быстрее, а объектно-ориентированное приложение на 50% быстрее.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выполнены все основные цели и задачи, а именно: закреплены знания по курсу "Объектно-ориентированное программирование" и приобретены навыки объектно-ориентированной и процедурной реализаций прикладной задачи (задачи вычисления площади геометрической фигуры методом Монте Карло) с использованием различных языков, инструментальных систем и библиотек, автоматизирующих проектирование, программирование и отладку создаваемых приложений.

Также были приобретены навыки решения вычислительных задач; практически освоены современные инструментальные системы разработки ПО; был проведён сравнительный анализ вычислительной эффективности процедурных и объектно-ориентированных программ; получены навыки создания программ на языках C++ и C# платформы Microsoft .NET Framework; приобретены практические навыки оформления и выпуска документации в соответствии с требованиями стандартов (ЕСПД, UML).

# СПИСОК ЛИТЕРАТУРЫ

1. Microsoft Docs [Электронный ресурс] : Документация по C#. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения 27.05.2020).
2. Metanit [Электронный ресурс] : Полное руководство по языку программирования С# 8.0 и платформе .NET Core 3. Режим доступа: https://metanit.com/sharp/tutorial/ (дата обращения 27.05.2020).
3. Wikipedia [Электронный ресурс] : Метод Монте-Карло. Режим доступа: https://ru.wikipedia.org/wiki/Метод\_Монте-Карло (дата обращения 27.05.2020).
4. Wikipedia [Электронный ресурс] : UML. Режим доступа: https://ru.wikipedia.org/wiki/ UML (дата обращения 27.05.2020).
5. Троелсен Э., Джепикс Ф. Язык программирования C# 7 и платформы. NET и. NET Core. – Litres, 2019.

# ПРИЛОЖЕНИЕ

## Файл Program.cs (процедурное приложение)

using System;

namespace Procedure

{

class Program

{

static bool IsPointInTriangle(Point p, Point b, Point e, Point o)

{

Point e\_0 = new Point(e.X - b.X, e.Y - b.Y), o\_0 = new Point(o.X - b.X, o.Y - b.Y), p\_0 = new Point(p.X - b.X, p.Y - b.Y);

double m = ( p\_0.X \* e\_0.Y - e\_0.X \* p\_0.Y ) / ( o\_0.X \* e\_0.Y - e\_0.X \* o\_0.Y );

if (m >= 0 && m <= 1)

{

double l = ( p\_0.X - m \* o\_0.X ) / e\_0.X;

if (l >= 0 && m + l <= 1)

{

return true;

}

}

return false;

}

static bool IsPointInCircle(Point p, Point o, double R)

{

return Math.Pow(p.X - o.X, 2) + Math.Pow(p.Y - o.Y, 2) <= Math.Pow(R, 2);

}

static void MonteCarlo(Point d, Point e, Point b, Point o)

{

double R = e.Y - o.Y;

double S = ( e.X - b.X ) \* ( e.Y - o.Y ) / 2 + Math.PI \* Math.Pow(R, 2) / 4, mcrS = ( d.X - b.X ) \* ( b.Y - d.Y );

Console.WriteLine($"Площадь фигуры debo равна {S}");

Console.WriteLine($"Площадь прямоугольника Монте-Карло равна {mcrS}");

Random rnd = new Random();

for (int N = 1000; N <= 10000000; N \*= 10)

{

int M = 0;

DateTime time = DateTime.Now;

for (int i = 0; i < N; ++i)

{

Point p = new Point(b.X + ( d.X - b.X ) \* rnd.NextDouble(), d.Y + ( b.Y - d.Y ) \* rnd.NextDouble());

if (p.X < o.X && IsPointInTriangle(p, b, e, o) || p.X == o.X && p.Y >= o.Y && p.Y <= e.Y || p.X > o.X && IsPointInCircle(p, o, R))

{

++M;

}

}

TimeSpan dt = DateTime.Now - time;

double mcS = mcrS \* M / N;

Console.WriteLine($"N = {N}:\tПлощадь фигуры debo примерно равна {mcS}.\tПогрешность вычислений - {Math.Abs(1 - mcS / S) \* 100}%.\tРезультат получен за {dt:s\\.FFFFFFF} с.");

}

}

static void Manual()

{

input: Console.WriteLine("Введите координаты точки d:");

Console.Write("x = ");

double x = Convert.ToDouble(Console.ReadLine());

Console.Write("y = ");

double y = Convert.ToDouble(Console.ReadLine());

Point d = new Point(x, y);

Console.WriteLine("Введите координаты точки b:");

Console.Write("x = ");

x = Convert.ToDouble(Console.ReadLine());

Console.Write("y = ");

y = Convert.ToDouble(Console.ReadLine());

Point b = new Point(x, y), e = new Point(d.X - b.Y + d.Y, b.Y), o = new Point(e.X, d.Y);

if (d.X <= b.X || b.Y <= d.Y || e.X < b.X)

{

Console.WriteLine("Ошибка! Недопустимое расположение точек b и d!");

goto input;

}

Console.WriteLine($"Координаты точки d:\nx = {d.X}\ny = {d.Y}");

Console.WriteLine($"Координаты точки e:\nx = {e.X}\ny = {e.Y}");

Console.WriteLine($"Координаты точки b:\nx = {b.X}\ny = {b.Y}");

Console.WriteLine($"Координаты точки o:\nx = {o.X}\ny = {o.Y}");

MonteCarlo(d, e, b, o);

}

static void Test()

{

Point d = new Point(10, 1), e = new Point(8, 3), b = new Point(1, 3), o = new Point(8, 1);

Console.WriteLine($"Координаты точки d:\nx = {d.X}\ny = {d.Y}");

Console.WriteLine($"Координаты точки e:\nx = {e.X}\ny = {e.Y}");

Console.WriteLine($"Координаты точки b:\nx = {b.X}\ny = {b.Y}");

Console.WriteLine($"Координаты точки o:\nx = {o.X}\ny = {o.Y}");

MonteCarlo(d, e, b, o);

}

static void Main()

{

string action = "";

while (action != "выход")

{

Console.WriteLine("Выберите действие:");

Console.WriteLine("[Ввод] - Ручной ввод координат\t[Тест] - Выполнить контрольный пример\t[Выход] - Выйти");

Console.Write("> ");

input: action = Console.ReadLine().ToLower();

switch (action)

{

case "ввод": Manual(); break;

case "тест": Test(); break;

case "выход": break;

default: Console.Write("Действие не найдено. Повторите ввод:\n> "); goto input;

}

}

}

}

}

## Файл Point.cs

namespace Procedure

{

public class Point

{

public double X { get; private set; }

public double Y { get; private set; }

public Point()

{

X = 0;

Y = 0;

}

public Point(double x, double y)

{

X = x;

Y = y;

}

public Point(Point p)

{

X = p.X;

Y = p.Y;

}

}

}

## Файл Program.cs (объектно-ориентированное приложение)

using System;

namespace ObjectOriented

{

class Program

{

static void Manual()

{

input: Console.WriteLine("Введите координаты точки d:");

Console.Write("x = ");

double x = Convert.ToDouble(Console.ReadLine());

Console.Write("y = ");

double y = Convert.ToDouble(Console.ReadLine());

Point d = new Point(x, y);

Console.WriteLine("Введите координаты точки b:");

Console.Write("x = ");

x = Convert.ToDouble(Console.ReadLine());

Console.Write("y = ");

y = Convert.ToDouble(Console.ReadLine());

Point b = new Point(x, y), e = new Point(d.X - b.Y + d.Y, b.Y), o = new Point(e.X, d.Y);

if (d.X <= b.X || b.Y <= d.Y || e.X < b.X)

{

Console.WriteLine("Ошибка! Недопустимое расположение точек b и d!");

goto input;

}

Console.WriteLine($"Координаты точки d:\nx = {d.X}\ny = {d.Y}");

Console.WriteLine($"Координаты точки e:\nx = {e.X}\ny = {e.Y}");

Console.WriteLine($"Координаты точки b:\nx = {b.X}\ny = {b.Y}");

Console.WriteLine($"Координаты точки o:\nx = {o.X}\ny = {o.Y}");

Figure f = new Figure(d, e, b, o);

f.MonteCarlo();

}

static void Test()

{

Point d = new Point(10, 1), e = new Point(8, 3), b = new Point(1, 3), o = new Point(8, 1);

Console.WriteLine($"Координаты точки d:\nx = {d.X}\ny = {d.Y}");

Console.WriteLine($"Координаты точки e:\nx = {e.X}\ny = {e.Y}");

Console.WriteLine($"Координаты точки b:\nx = {b.X}\ny = {b.Y}");

Console.WriteLine($"Координаты точки o:\nx = {o.X}\ny = {o.Y}");

Figure f = new Figure(d, e, b, o);

f.MonteCarlo();

}

static void Main()

{

string action = "";

while (action != "выход")

{

Console.WriteLine("Выберите действие:");

Console.WriteLine("[Ввод] - Ручной ввод координат\t[Тест] - Выполнить контрольный пример\t[Выход] - Выйти");

Console.Write("> ");

input: action = Console.ReadLine().ToLower();

switch (action)

{

case "ввод": Manual(); break;

case "тест": Test(); break;

case "выход": break;

default: Console.Write("Действие не найдено. Повторите ввод:\n> "); goto input;

}

}

}

}

}

## Файл Point.cs

using System;

namespace ObjectOriented

{

class Rect : Point

{

public double DX { get; set; }

public double DY { get; set; }

public Rect() : base()

{

DX = 1;

DY = 1;

}

public Rect(Point p0, Point p) : base(Math.Min(p0.X, p.X), Math.Min(p0.Y, p.Y))

{

DX = Math.Abs(p.X - p0.X);

DY = Math.Abs(p.Y - p0.Y);

}

public Rect(Rect r) : base(r.X, r.Y)

{

DX = r.DX;

DY = r.DY;

}

public override string ToString()

{

return $"[{base.ToString()}, {DX}, {DY}]";

}

public double Square()

{

return DX \* DY;

}

}

}

## Файл Figure.cs

using System;

namespace ObjectOriented

{

class Figure

{

private readonly Circle c;

private readonly Triangle t;

private readonly Rect r;

public Figure()

{

c = new Circle();

t = new Triangle();

r = new Rect();

}

public Figure(Point d, Point e, Point b, Point o)

{

c = new Circle(o, e.Y - o.Y);

t = new Triangle(b, e, o);

r = new Rect(b, d);

}

public Figure(Figure f)

{

c = f.c;

t = f.t;

r = f.r;

}

public double Square()

{

return c.Square() / 4 + t.Square();

}

public bool IsPointHere(Point p)

{

return p.X < t.O.X ? t.IsPointHere(p) : c.IsPointHere(p);

}

public void MonteCarlo()

{

double S = Square(), mcrS = r.Square();

Console.WriteLine($"Площадь фигуры debo равна {S}");

Console.WriteLine($"Площадь прямоугольника Монте-Карло равна {mcrS}");

Random rnd = new Random();

for (int N = 1000; N <= 10000000; N \*= 10)

{

int M = 0;

DateTime time = DateTime.Now;

for (int i = 0; i < N; ++i)

{

Point p = new Point(r.X + r.DX \* rnd.NextDouble(), r.Y + r.DY \* rnd.NextDouble());

if (IsPointHere(p))

{

++M;

}

}

TimeSpan dt = DateTime.Now - time;

double mcS = mcrS \* M / N;

Console.WriteLine($"N = {N}:\tПлощадь фигуры debo примерно равна {mcS}.\tПогрешность вычислений - {Math.Abs(1 - mcS / S) \* 100}%.\tРезультат получен за {dt:s\\.FFFFFFF} с.");

}

}

}

}

## Файл Point.cs

namespace ObjectOriented

{

public class Point

{

public double X { get; private set; }

public double Y { get; private set; }

public Point()

{

X = 0;

Y = 0;

}

public Point(double x, double y)

{

X = x;

Y = y;

}

public Point(Point p)

{

X = p.X;

Y = p.Y;

}

}

}

## Файл Circle.cs

using System;

namespace ObjectOriented

{

internal class Circle : Point

{

public double R { get; private set; }

public Circle() : base()

{

R = 1;

}

public Circle(Point p, double r) : base(p)

{

R = r;

}

public Circle(Circle c) : base(c.X, c.Y)

{

R = c.R;

}

public double Square()

{

return Math.PI \* Math.Pow(R, 2);

}

public bool IsPointHere(Point p)

{

return Math.Pow(p.X - X, 2) + Math.Pow(p.Y - Y, 2) <= Math.Pow(R, 2);

}

}

}

## Файл Triangle.cs

using System;

namespace ObjectOriented

{

class Rect : Point

{

public double DX { get; private set; }

public double DY { get; private set; }

public Rect() : base()

{

DX = 1;

DY = 1;

}

public Rect(Point p0, Point p) : base(Math.Min(p0.X, p.X), Math.Min(p0.Y, p.Y))

{

DX = Math.Abs(p.X - p0.X);

DY = Math.Abs(p.Y - p0.Y);

}

public Rect(Rect r) : base(r.X, r.Y)

{

DX = r.DX;

DY = r.DY;

}

public double Square()

{

return DX \* DY;

}

}

}

## Файл Triangle.cs

using System;

namespace ObjectOriented

{

class Triangle

{

public Point B { get; private set; }

public Point E { get; private set; }

public Point O { get; private set; }

public Triangle()

{

B = new Point(0, 0);

E = new Point(1, 0);

O = new Point(0, 1);

}

public Triangle(Point b, Point e, Point o)

{

B = b;

E = e;

O = o;

}

public Triangle(Triangle t)

{

B = t.B;

E = t.E;

O = t.O;

}

public double Square()

{

return ( E.X - B.X ) \* ( E.Y - O.Y ) / 2;

}

public bool IsPointHere(Point p)

{

Point e\_0 = new Point(E.X - B.X, E.Y - B.Y), o\_0 = new Point(O.X - B.X, O.Y - B.Y), p\_0 = new Point(p.X - B.X, p.Y - B.Y);

double m = ( p\_0.X \* e\_0.Y - e\_0.X \* p\_0.Y ) / ( o\_0.X \* e\_0.Y - e\_0.X \* o\_0.Y );

if (m >= 0 && m <= 1)

{

double l = ( p\_0.X - m \* o\_0.X ) / e\_0.X;

if (l >= 0 && m + l <= 1)

{

return true;

}

}

return false;

}

}

}