Оглавление

[Модульное тестирование в Visual Studio 1](#_Toc79524835)

[Создание проекта программы, модули которой будут тестироваться 1](#_Toc79524836)

[Создание проекта теста в Visual Studio 1](#_Toc79524837)

[Задание 8](#_Toc79524838)

[Дополнительное задание 9](#_Toc79524839)

# Модульное тестирование в Visual Studio

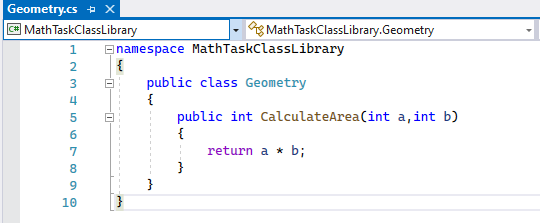
Модульное тестирование (Unit-тестирование) предназначено для проверки правильности выполнения небольшого блока кода, решающего свою конкретную задачу. В работе показано, как проводить в модульное тестирование в Visual Studio.

## Создание проекта программы, модули которой будут тестироваться

Разработаем проект содержащий класс, который вычисляет площадь прямоугольника по длине двух его сторон.

Создадим в Visual Studio новый проект Visual C# -> Библиотека классов. Назовём его **MathTaskClassLibrary**.

Создадим в нем класс **Geometry**, в котором реализуем метод, вычисляющий площадь прямоугольника. Для демонстрации остановимся на работе с целыми числами. Код программы приведён ниже.

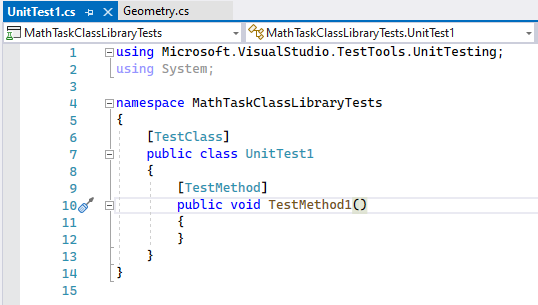


## Создание проекта теста в Visual Studio

Чтобы выполнить unit-тестирование, необходимо в рамках того же самого решения создать ещё один проект соответствующего типа. Правой кнопкой щёлкните по решению, выберите “Добавить” и затем “Создать проект…”.

В открывшемся окне в группе Visual C# щёлкните “Тест”, а затем выберите “Проект модульного теста”. Введите имя проекта **MathTaskClassLibraryTests**и нажмите “ОК”. Таким образом проект будет создан.

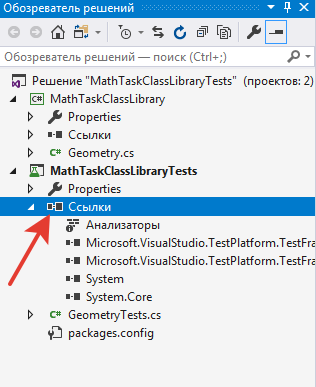
Перед Вами появится следующий код:



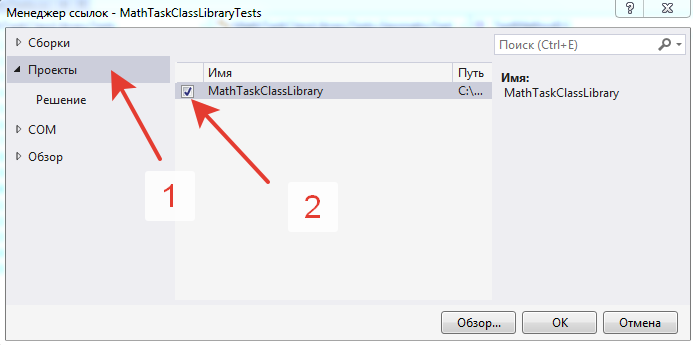
Атрибут [TestMethod] означает, что далее идёт метод, содержащий модульный (unit) тест, [TestClass] в свою очередь говорит о том, что далее идёт класс, содержащий методы, в которых присутствуют unit-тесты.

В соответствии с принятыми соглашениями переименуем класс UnitTest1 в **GeometryTests**.

Затем в ссылках проекта (References) необходимо добавить ссылку на проект, код которого будем тестировать. Правой кнопкой щёлкаем на References, а затем выбираем “Добавить ссылку…”:



В появившемся выбираем “Проекты” и ставим галочку напротив проекта MathTaskClassLibrary. Затем жмём “ОК”.



Также в коде файла теста необходимо подключить с помощью директивы using следующее пространство имён:

using MathTaskClassLibrary;

Займёмся написание теста. Проверим поведение программы для следующих исходных данных:

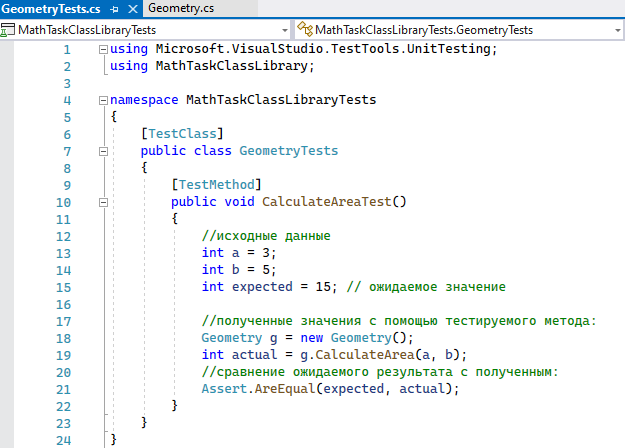
1. Стороны прямоугольника составляют 3 и 5. Ожидаемый результат (правильное решение) в данном случае - число 15.
2. Одна из сторон равна отрицательному числу, например -4, другая – произвольному, например, 10. Поведение функции расчета площади на таких данных не определено, должно быть сгенерировано исключение типа System.ArgumentException. Соответственно, тест будет пройден, если будет поймано исключение такого типа.

Таким образом, тест должен проверять поведение системы не только внутри диапазона допустимых значений, но также на границах и за его пределами.

Переименуем метод TestMethod1() в **CalculateAreaTest ()**. Тестирующий метод обычно содержит три необходимых компонента:

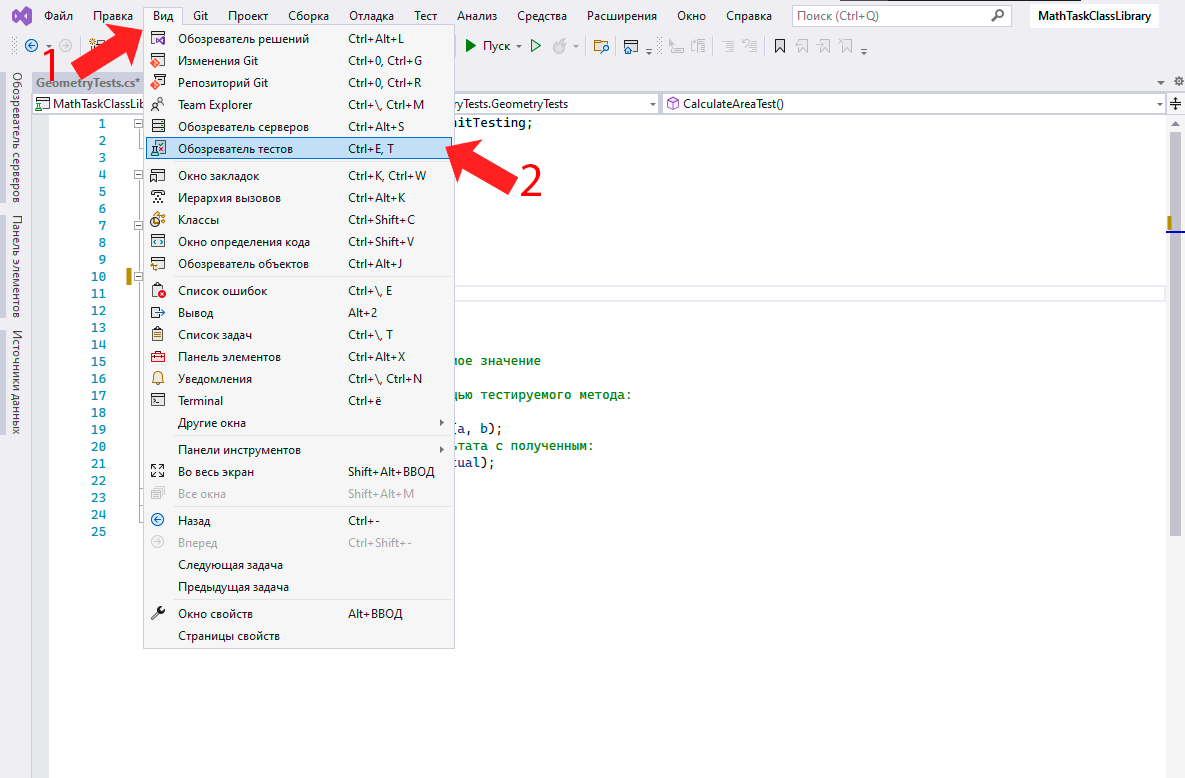
1. исходные данные: входные значения и ожидаемый результат;
2. код, вычисляющий значение с помощью тестируемого метода;
3. код, сравнивающий ожидаемый результат с полученным.

Для первого случая тестирующий код будет следующим:

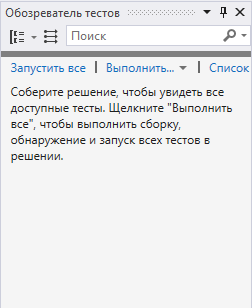


Для сравнения ожидаемого результата с полученным используется метод **AreEqual** класса **Assert**. Данный класс всегда используется при написании unit тестов в Visual Studio.

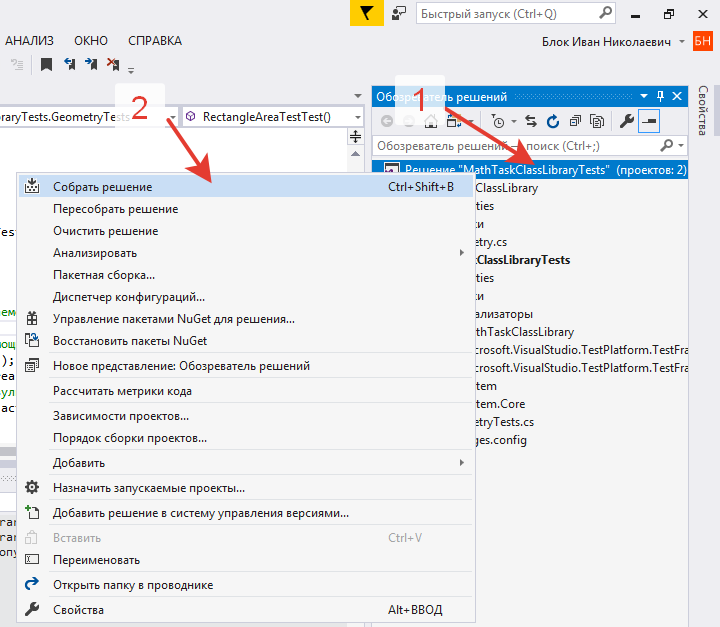
Теперь, чтобы просмотреть все тесты, доступные для выполнения, необходимо открыть окно “Обозреватель тестов”. Для этого в меню Visual Studio щёлкните на кнопку “Вид”, а затем нажмите на пункт “Обозреватель тестов”.

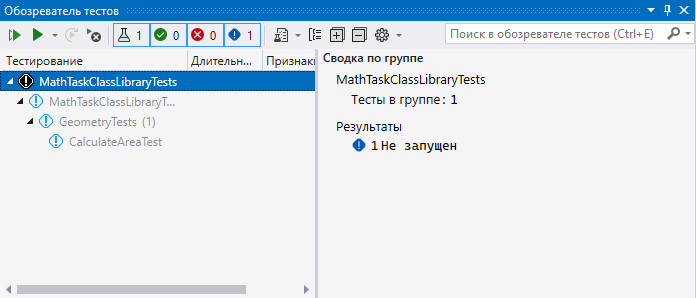


Будет открыто следующее окно:

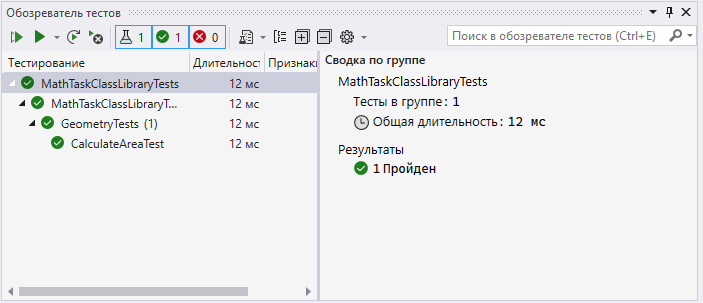


В данный момент список тестов пуст, поскольку решение ещё ни разу не было собрано. Выполним построение решения, кликнув правой кнопкой по решению в обозревателе решений и выбрав пункт «Собрать решения»:

 После её завершения в “Обозревателе тестов” появится наш тест:



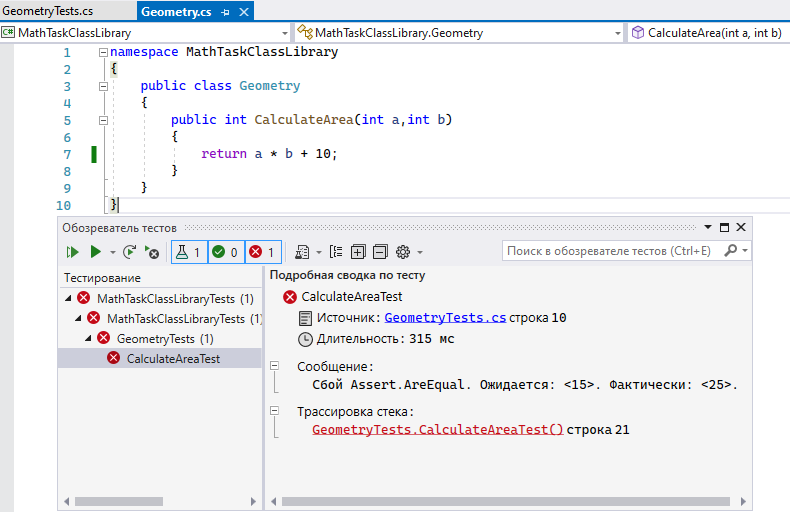
Синяя табличка с восклицательным знаком означает, что указанный тест никогда не выполнялся. Выполним его, для этого нажмём правой кнопкой мыши на его имени и выберем “Выполнить выбранные тесты”.



Зелёный кружок с галочкой означает, что тест успешно пройден: ожидаемый и полученный результаты совпадают.

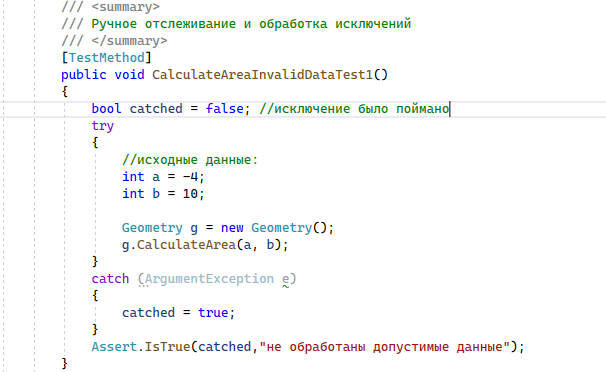
Изменим код метода CalculateArea, вычисляющего площадь прямоугольника, чтобы сымитировать провал теста и посмотреть, как поведёт себя Visual Studio. Прибавим к возвращаемому значению 10.

Запустим unit-тест.



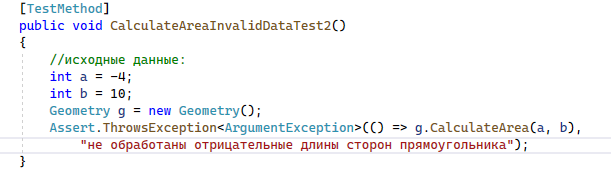
Как видно, тест не пройден, а в сообщении ниже указаны причина провала теста и его местоположение.

Обработаем вторую ситуацию, когда одна из сторон является отрицательным числом. Тест может считаться успешно пройденным в том случае, если в тестируемой функции было сгенерировано исключение. Реализовать данный тест можно двумя способами: вручную отлавливая исключение с помощью конструкции try catch, либо используя метод Assert.ThrowsException, рассмотрим оба. Для этого добавим методы CalculateAreaInvalidDataTest1 и CalculateAreaInvalidDataTest2.

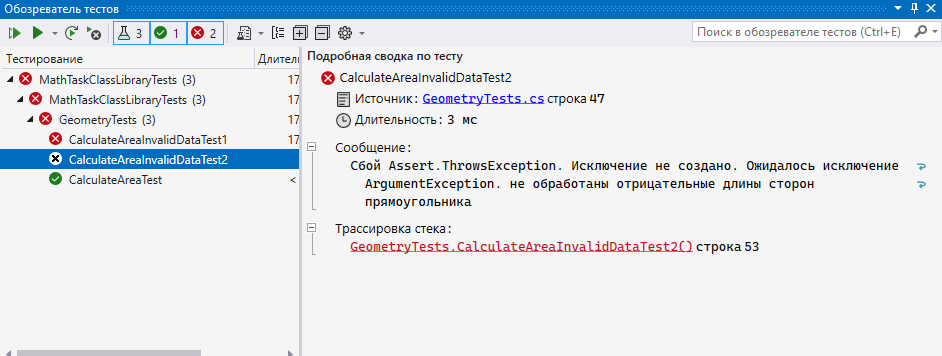


Для данной проверки нам не нужно сравнивать ожидаемое значение с полученным, т.к. ожидаемый результат не определен, поэтому мы просто вызываем расчет площади и ожидаем генерации исключения.

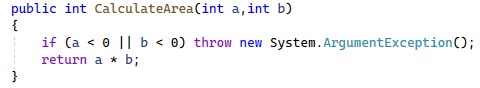
Второй способ выполняет ту же функцию, но позволяет сократить объем кода:



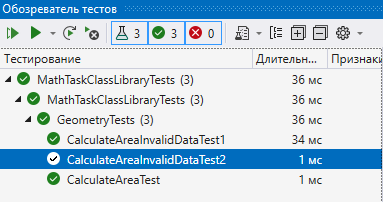
Запустим полученные тесты:



Поскольку метод CalculateArea класса Geometry не обрабатывает ошибки такого рода, мы получили 2 провальных теста. Чтобы тесты были выполнены, необходимо привести код CalculateArea к следующему виду:



После перезапуска получаем 3 пройденных теста из 3х:

.

## Задание

Проверьте работу нижеприведенных функций с помощью модульного тестирования. При тестировании следует брать несколько значений – из интервала допустимых значений, а также параметры, лежащие вне допустимого интервала. При передаче заведомо недопустимых параметров, тестируемый метод должен генерировать исключение. Тест в этом случаем, считается успешным, если было сгенерировано исключение типа ArgumentException (или исключение – наследник ArgumentException).

Для каждого пункта задания должен быть создан отдельный тестирующий класс, для каждого проверяемого значения – отдельный тестовый метод в этом классе.

1. Дано целое число N (1 ≤ N ≤ 26). Сформировать строку, содержащую N первых прописных (т.е. заглавных) букв латинского алфавита.
2. Решить квадратное уравнения, заданное коэффициентами вида ax2+bx+c = 0, где a,b,c – некоторые вещественные коэффициенты. Метод должен возвращать массив корней соответствующего размера: 2 (D>0), 1 (D=0), 0 (D<0). Если a=0, должно генерироваться исключение ArgumentException, т.к. решение квадратного уравнения не определено. Тестирование производить с точностью 0.01.
3. Дан номер года (положительное целое число). Определить количество дней в этом году, учитывая, что обычный год насчитывает 365 дней, а високосный -366 дней. Високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300, и 1900 не являются високосными, а 1200 и 2000 – являются).
4. Написать регулярное выражение, которые проверят емейлы простого вида (например, [test@test.test](mailto:test@test.test)).
5. Дана строка, изображающая целое положительное число. Вывести сумму цифр этого числа.

## Дополнительное задание

1. Для лабораторной работы №6 протестируйте функцию интегрирования. Интегрирование должно выполняться с точностью до 0.01.
2. Для любых двух задач заполните шаблон «Лабораторная работа 7 - Отчет по тестированию.doc»