Оглавление

[1. СОЗДАНИЕ И ЗАПУСК МОДУЛЬНЫХ ТЕСТОВ ДЛЯ УПРАВЛЯЕМОГО КОДА 3](#_Toc152511776)

[1.1. Создание проекта для тестирования 3](#_Toc152511777)

[1.2. Создание проекта модульного теста, тестового класса и метода теста 4](#_Toc152511778)

[1.3. Сборка и запуск тестов 5](#_Toc152511779)

[1.4. Анализ результатов модульного теста и тестируемого кода 5](#_Toc152511780)

[2. СОЗДАНИЕ МОДУЛЬНОГО ТЕСТА, УПРАВЛЯЕМОГО ДАННЫМИ 6](#_Toc152511781)

[2.1. Создание тестируемого метода и метода теста 6](#_Toc152511782)

[2.2. Анализ покрытия кода тестами 8](#_Toc152511783)

[3. СОЗДАНИЕ, ИЗМЕНЕНИЕ И ОБСЛУЖИВАНИЕ ЗАКОДИРОВАННОО ТЕСТА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА 10](#_Toc152511784)

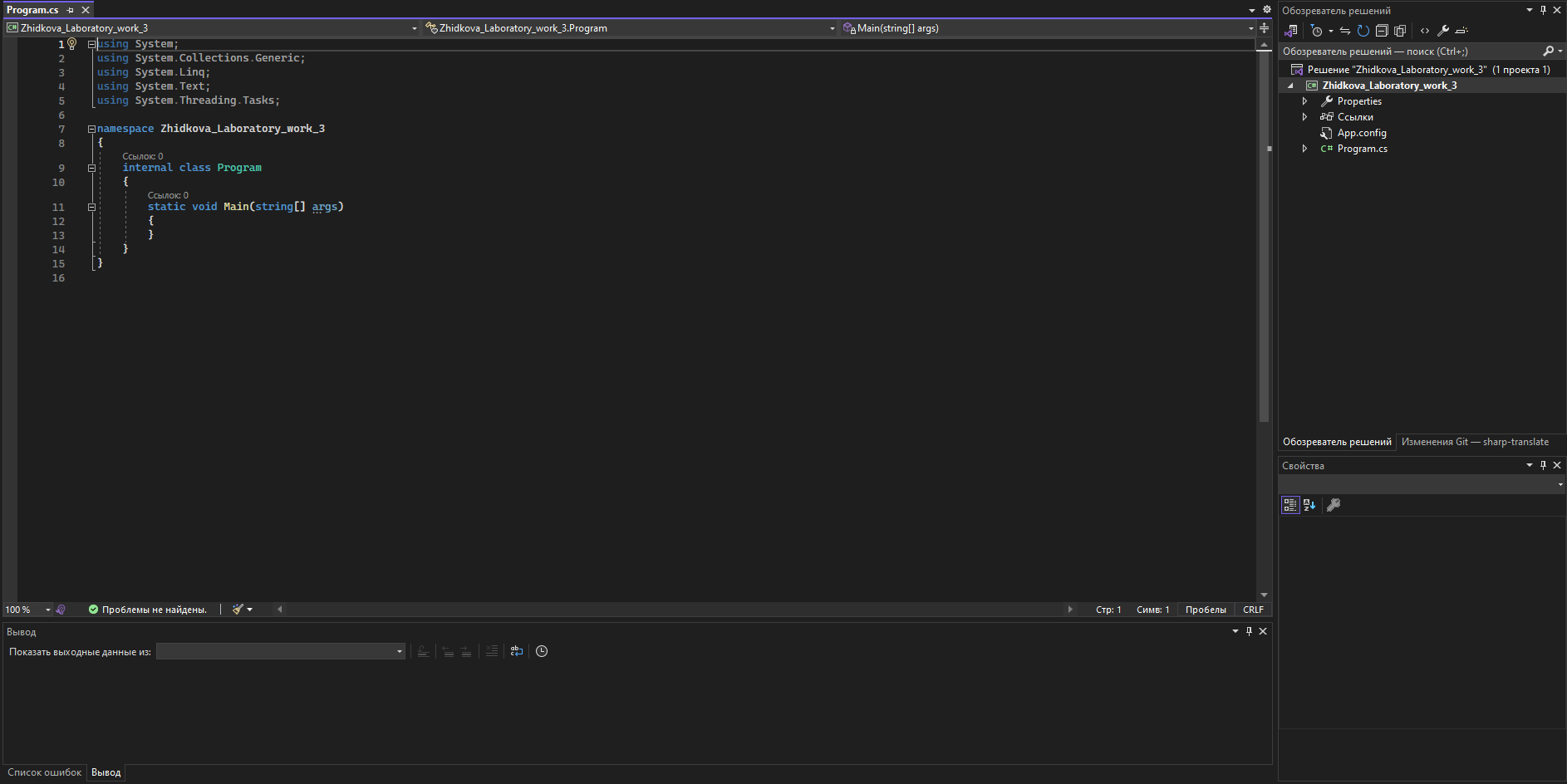
[3.1. Создание приложения WPF 10](#_Toc152511785)

[3.2. Создание ярлыка для приложения WPF и закодированные теста пользовательского интерфейса 11](#_Toc152511786)

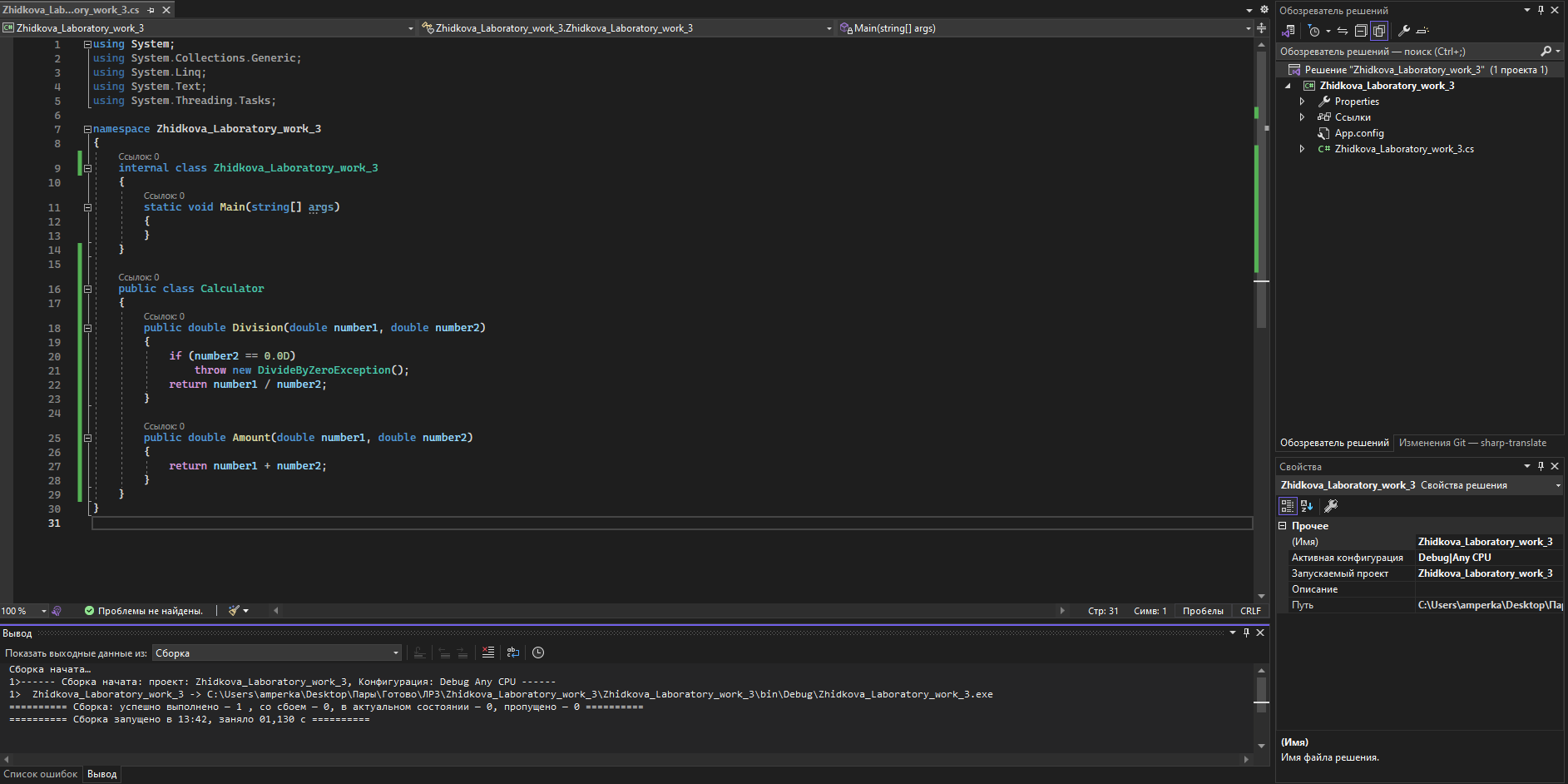
# СОЗДАНИЕ И ЗАПУСК МОДУЛЬНЫХ ТЕСТОВ ДЛЯ УПРАВЛЯЕМОГО КОДА

## Создание проекта для тестирования

Для проведения тестирования создаю проект «Zhidkova\_Laboratory\_work\_3». Проект отобразился в обозревателе решений, а основной файл Program.cs открылся в редакторе кода, как представлено ниже (рис. 1).

  
Рисунок 1. Обозреватель решения и редактор кода

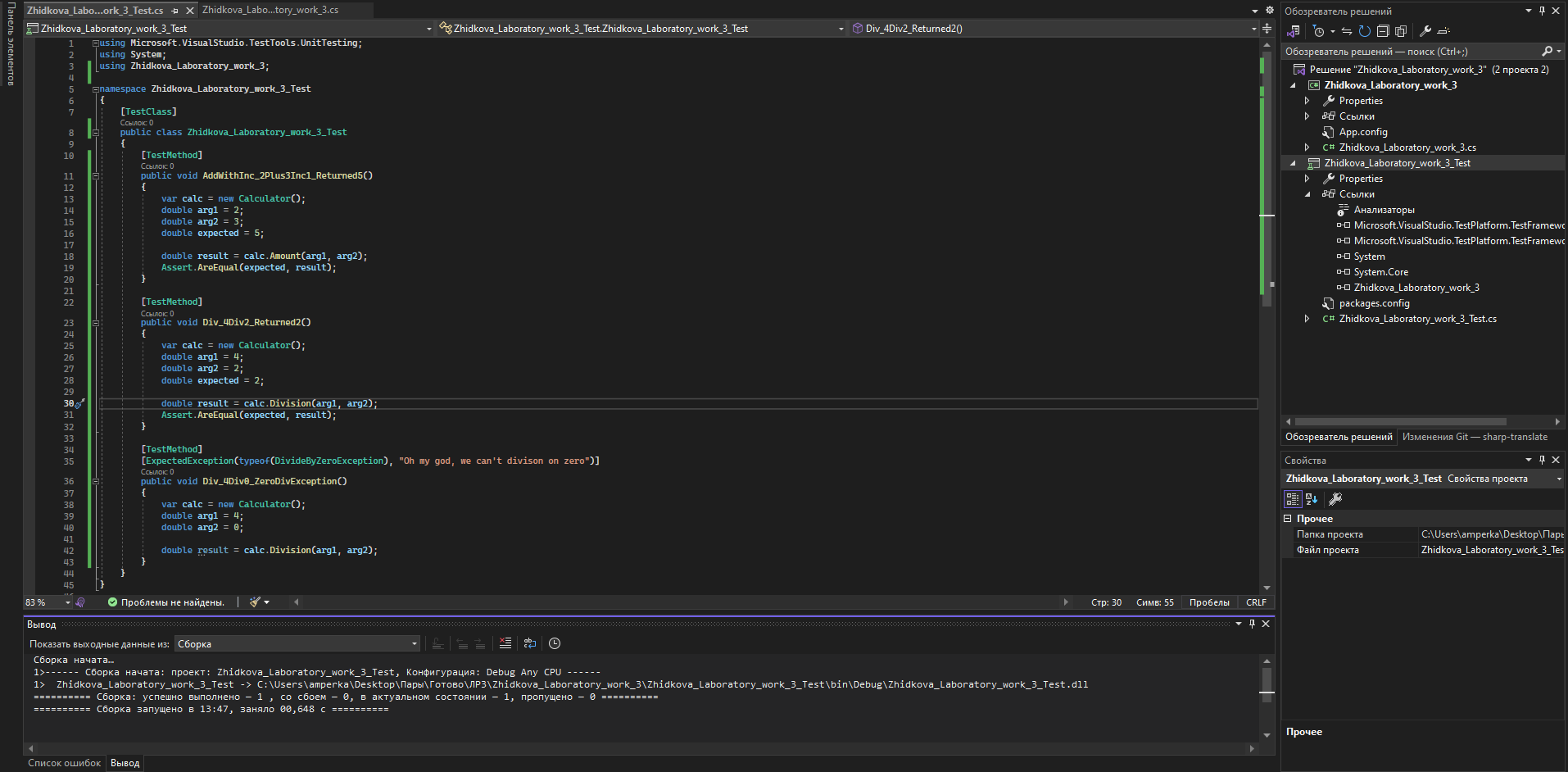
Основной файл проекта **Program.cs** переименую в **ZhidkovaCalculator.cs** и наполню содержимым основной файл проекта (рис. 2).

  
Рисунок 2. Основной файл проекта переименован и написан код

Таким образом получился проект с методами для тестирования. Тестирование будет проводиться на методах **Division** и **Amount**.

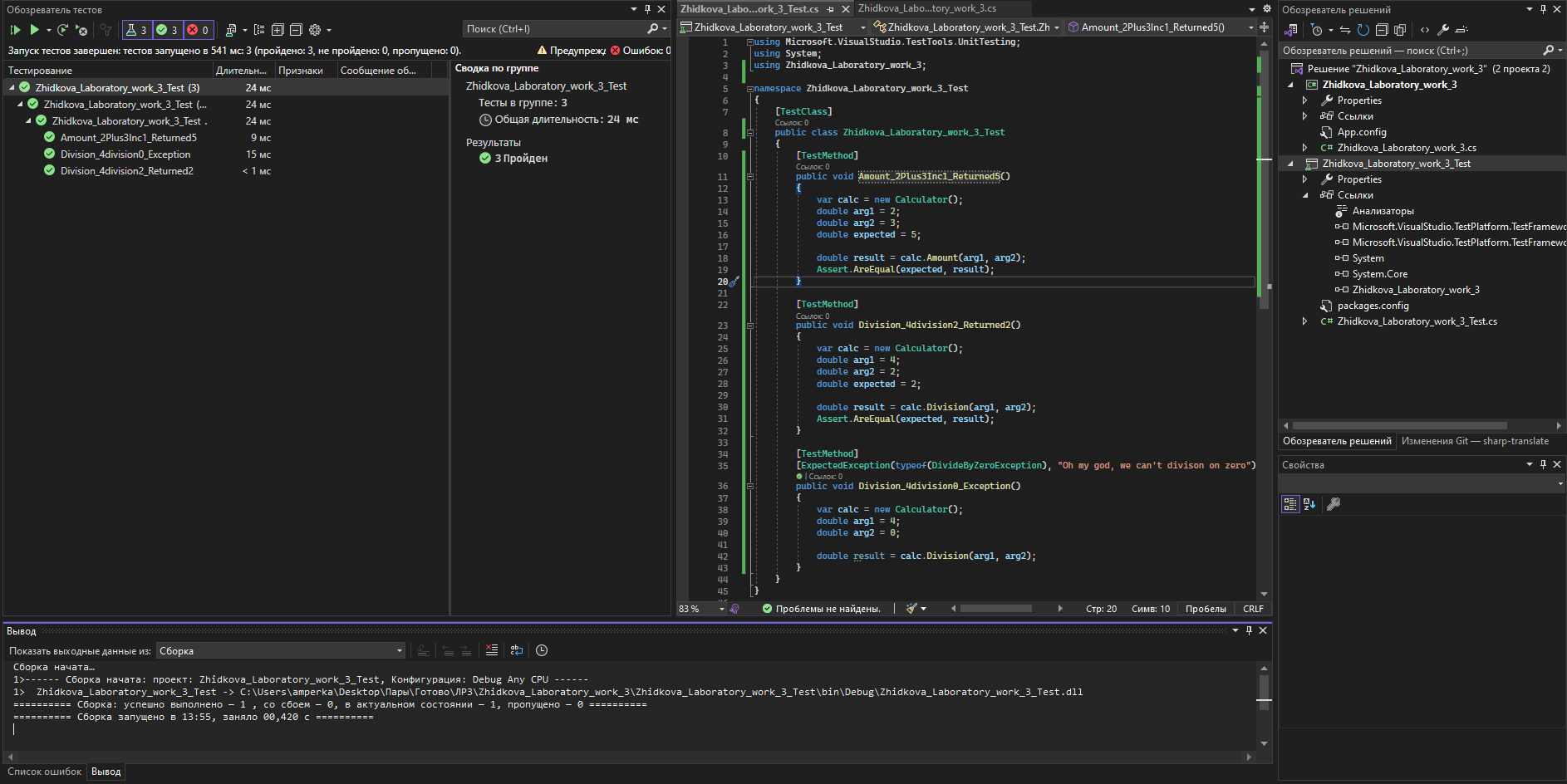
## Создание проекта модульного теста, тестового класса и метода теста

Для реализации тестирования в проект был добавлен модульный тест ‑ **Zhidkova\_Laboratory\_work\_3\_Test**. В модульный тест добавлена ссылка на основной проект **Zhidkova\_Laboratory\_work\_3**.

  
Рисунок 3. Модульный тест для методов основного проекта

## Сборка и запуск тестов

Подготовив основной проект и тесты методов проекта, необходимо собрать проект и запустить тестирование. В случае если в основном коде допущена ошибка, то тесты выведут покажут место с ошибкой. Если в основном коде отсутствуют ошибки, то тесты будут пройдены без предупреждений и ошибок (рис. 4).

  
Рисунок 4. Результат выполнения тестов

## Анализ результатов модульного теста и тестируемого кода

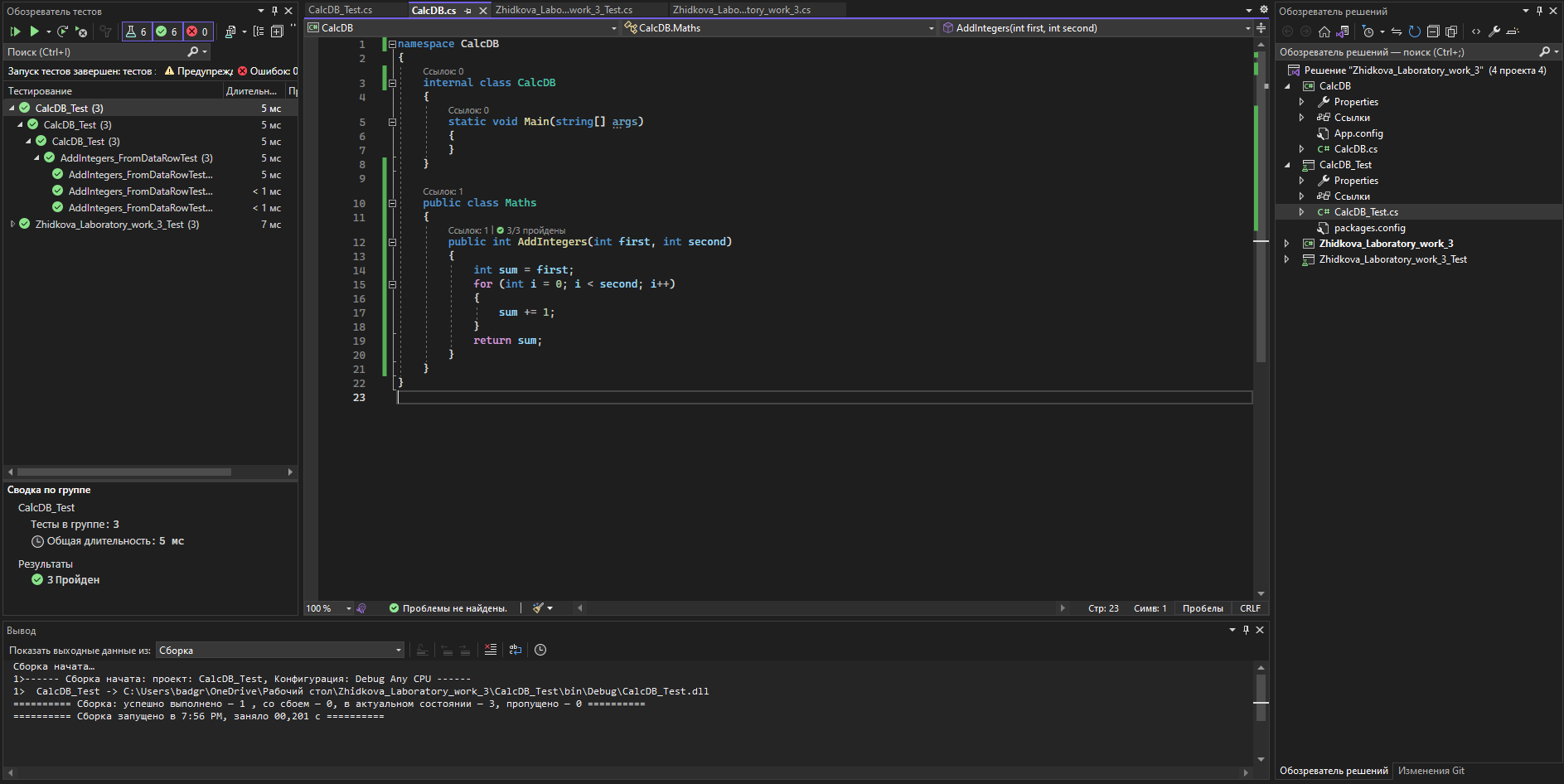
Так как при написание основного кода была предусмотрена защита от деления на нуль, то основной код можно считать надёжным. Основной способом повышения отказоустойчивости кода, написанного на C#, является метод обработки исключений. Поэтому при написании основного кода в методе **Division** была предусмотрена одна исключительная ситуация ‑ деление на нуль.

Для того чтобы обработать исключение было добавлено простое условие, которое проверяет равен делитель **number2** нулю или нет. Если делитель равен нулю, то вызывается экземпляр **DivideByZeroException** класса **Exception**.

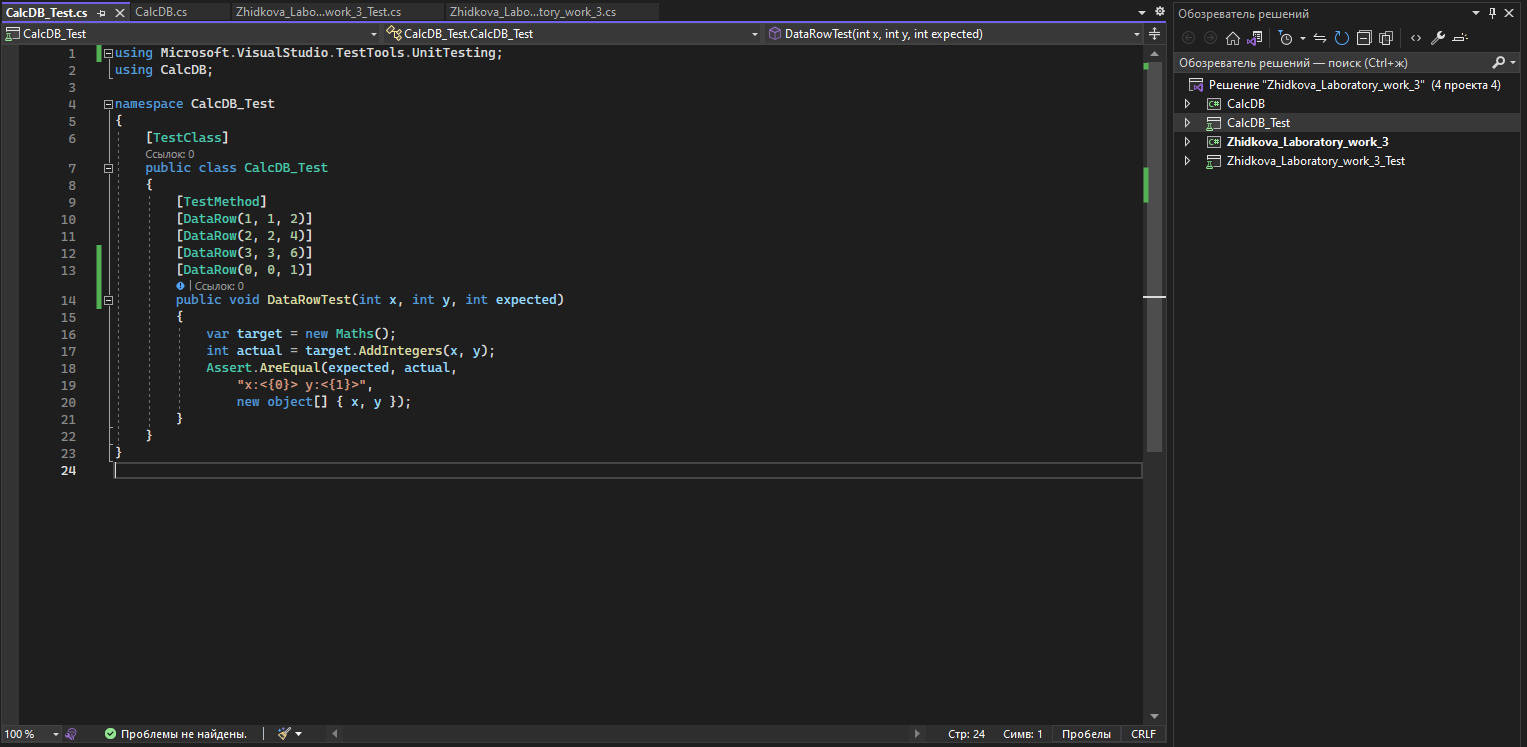
# СОЗДАНИЕ МОДУЛЬНОГО ТЕСТА, УПРАВЛЯЕМОГО ДАННЫМИ

## Создание тестируемого метода и метода теста

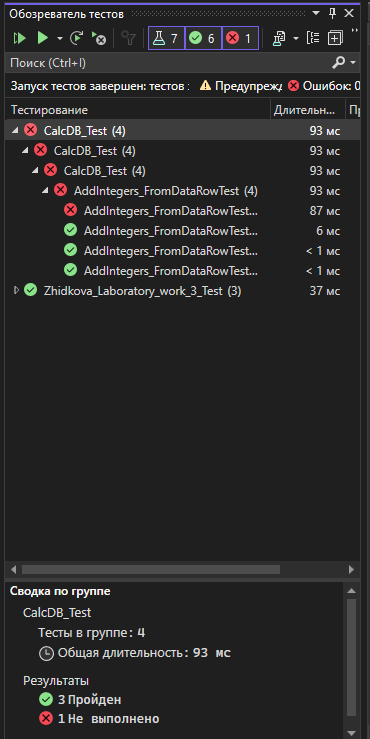
Для создания теста, управляемого данными, необходимо подготовить проект внутри решения, который будет носить название «CalcDB». Проект будет состоять из одного метода, который циклично считает сумму двух целых чисел (рис. 5).

  
Рисунок 5. Метод прокета «CalcDB»

Для того чтобы проверить функциональность метода класса был подготовлен модульный тест с именем **CalcDB\_Test**, который содержит метод под названием **DataRowTest** (рис. 6).

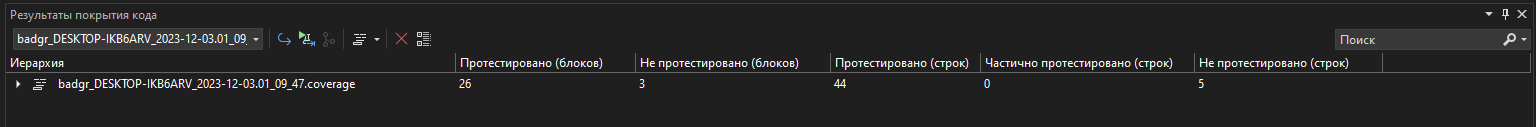
  
Рисунок 6. Метод DataRowTest модульного теста CalcDB\_Test

После успешной сборки решения было запущено тестирование, из четырёх заявленных тестов успешно завершены три (рис. 7). Это связанно с тем, что последний тест передаёт ошибочные данные.

  
Рисунок 7. Результато выполнения тестов

## Анализ покрытия кода тестами

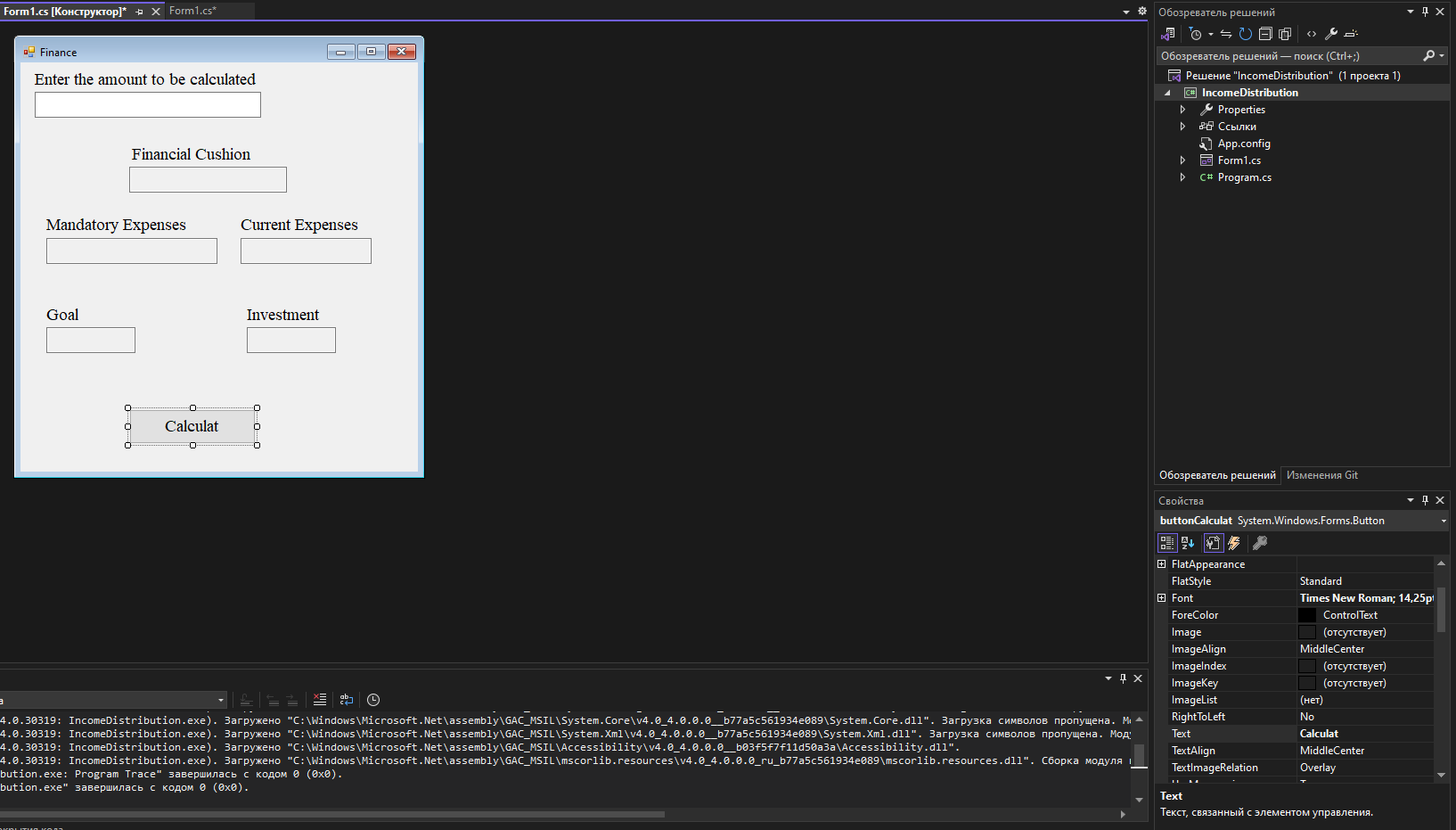
Исходя из анализа покрытия кода тестами – не протестированным остались системные строки кода (такие как инициализация библиотек), в остальном код протестирован полностью и тесты покрывают весь объём методов проекта.

  
Рисунок 8. Анализ покрытия кода тестами

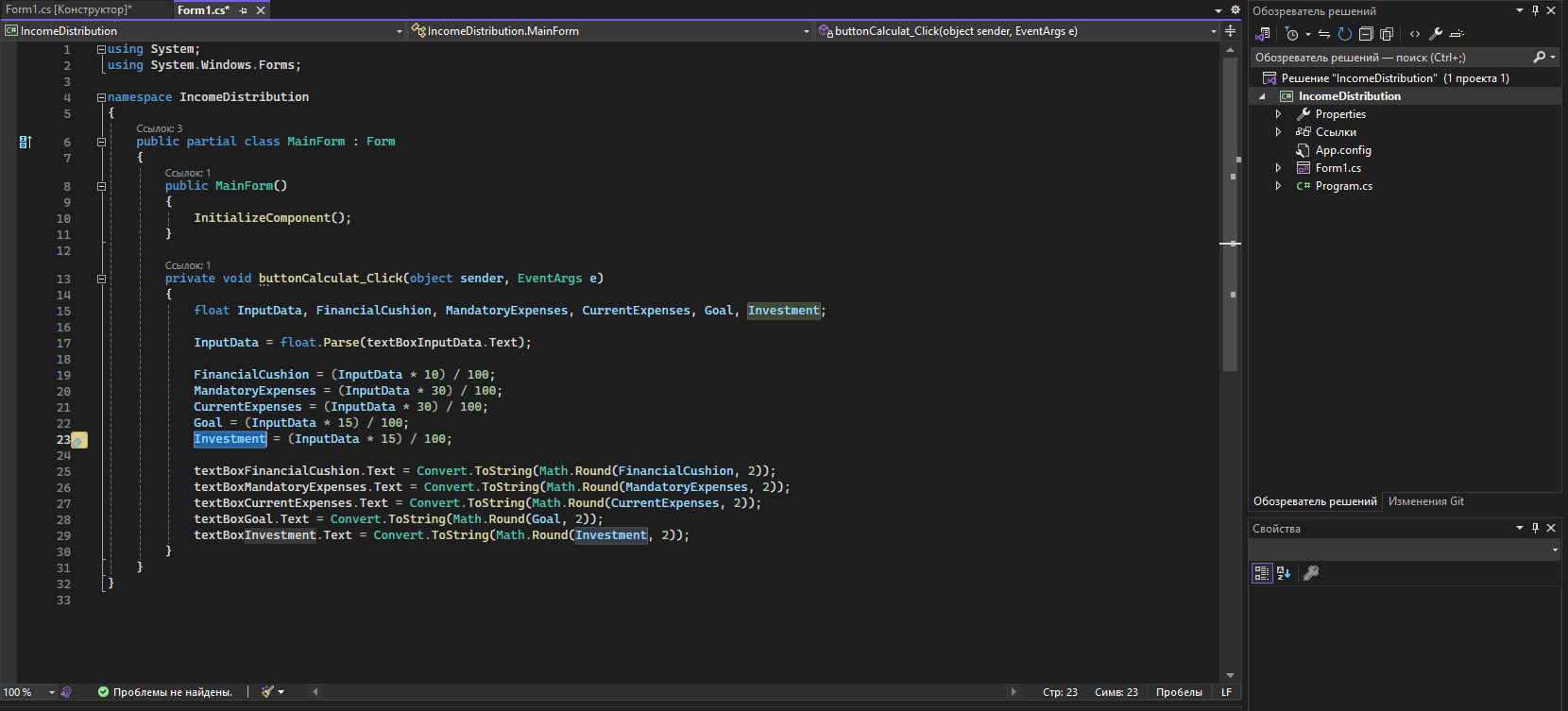
# СОЗДАНИЕ, ИЗМЕНЕНИЕ И ОБСЛУЖИВАНИЕ ЗАКОДИРОВАННОО ТЕСТА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

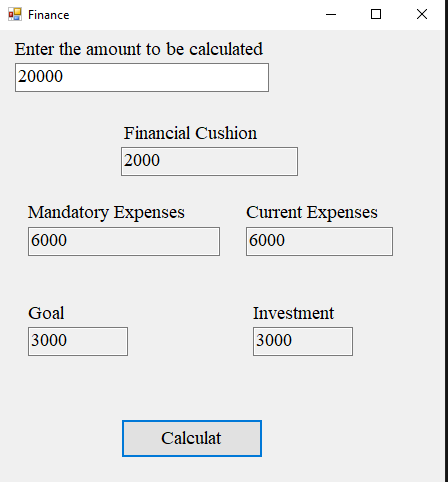
## Создание приложения WPF

В уже существующем решение создаю ещё один проект и назову его **finance** (рис. 9). В форму приложения перенесу основные элементы пользовательского интерфейса (button, TextBox, label)

  
Рисунок 9. Пользовательский интерфейс

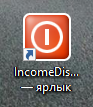
Для кнопки создаю обработчик одинарного клика по кнопке (рис. 10).

  
Рисунок 10. Обраюотчик нажатия кнопки

  
Рисунок 11. Результат работы программы

## Создание ярлыка для приложения WPF и закодированные теста пользовательского интерфейса

Для создания ярлыка приложения **IncomeDistribution** на рабочем столе необходимо было использовать команду копирования и вставки ярлыка, итог которого показан на скриншоте ниже (рис. 12).

  
Рисунок 12. Ярлык приложения