Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования

“Российский экономический университет имени Г.В. Плеханова”

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ**

Специальность: 09.02.07 Информационные системы и программирование  
Квалификация: программист

ПРАКТИЧЕСКИЕ РАБОТЫ ПО ДИСЦИПЛИНЕ

«ПОДДЕРЖКА И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ»

Листов: 12

|  |  |
| --- | --- |
| Выполнил студент  Группы: П50-4-21  Игошев Ростислав Вадимович | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_М. А. Горбунова  «\_\_\_\_\_\_» \_\_\_\_\_\_2023 года |

Москва 2023

ПРАКТИЧЕСКАЯ РАБОТА №1

Тема: «Unit-тесты пароля»

Цель работы: ознакомиться с unit-тестами на языке C#, научиться их писать, выполнить для примера метод проверки пароля и unit-тесты для него.

Начнём с создания проекта. Создадим проект под названием PasswordValidation и класс внутри него.

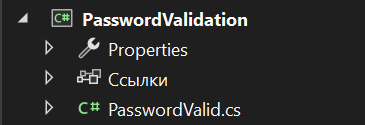


Рисунок 1 – Проект функционала

Теперь пропишем код внутри класса, создав метод с функционалом проверки пароля.

Условия для пароля (за каждое +балл):

1. Наличие цифр

2. Наличие строчных букв

3. Наличие заглавных букв

4. наличие спец. Символов

5. Содержать более 7 символов

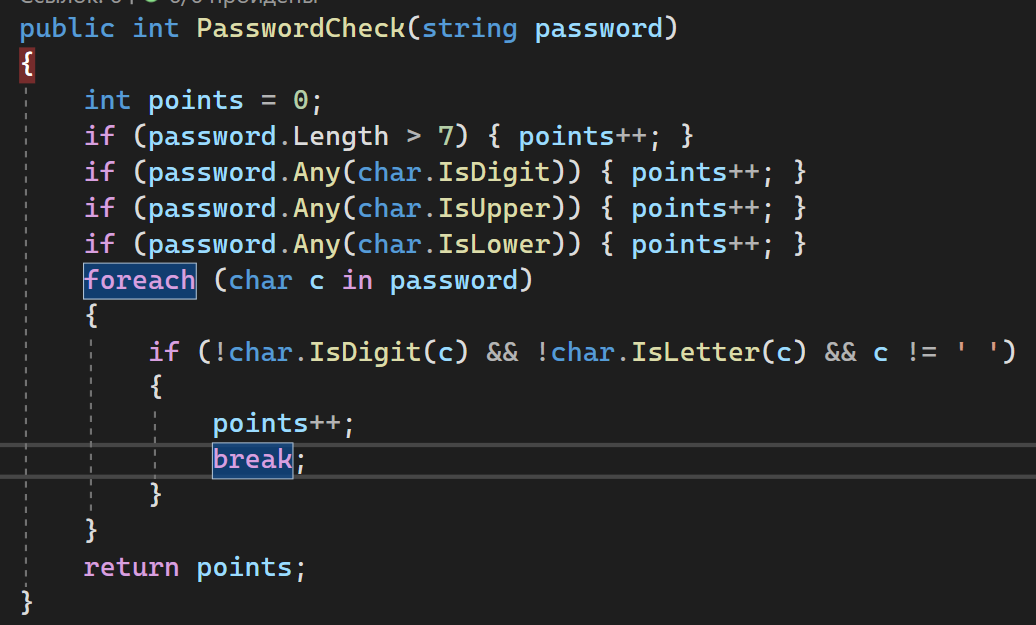


Рисунок 2 – Код функции проверки пароля

Теперь создадим проект тестирования нашей функции.

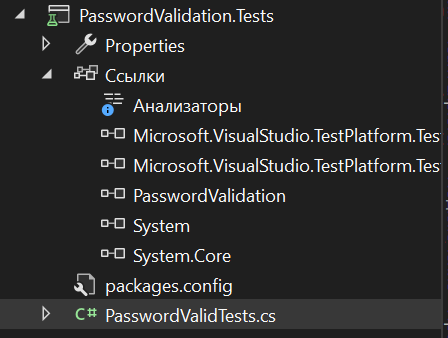


Рисунок 3 – Проект тестирования

Внутри проекта мы также добавим ссылку на тестируемый проект для обращения к методам внутри него.

Теперь создадим сами тесты. Выглядят они следующим образом (всего их 6):

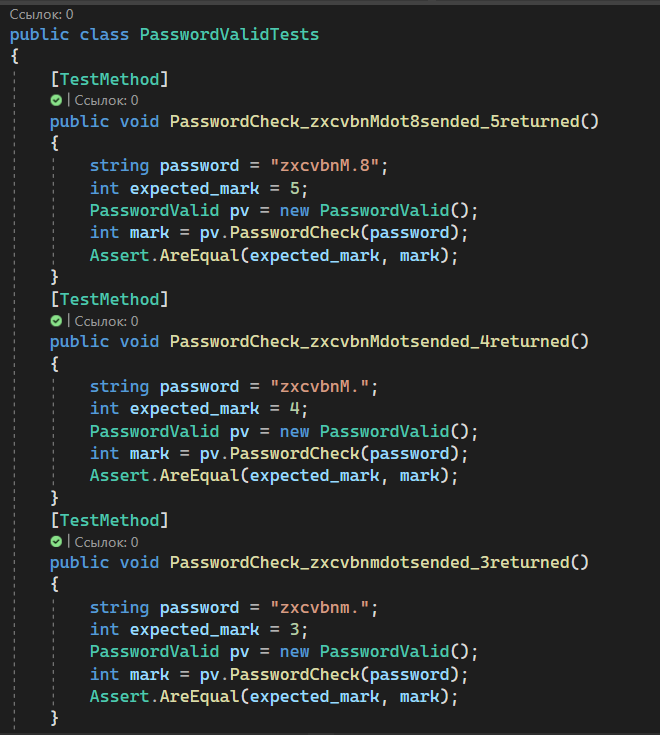


Рисунок 4 – Первые 3 метода тестов

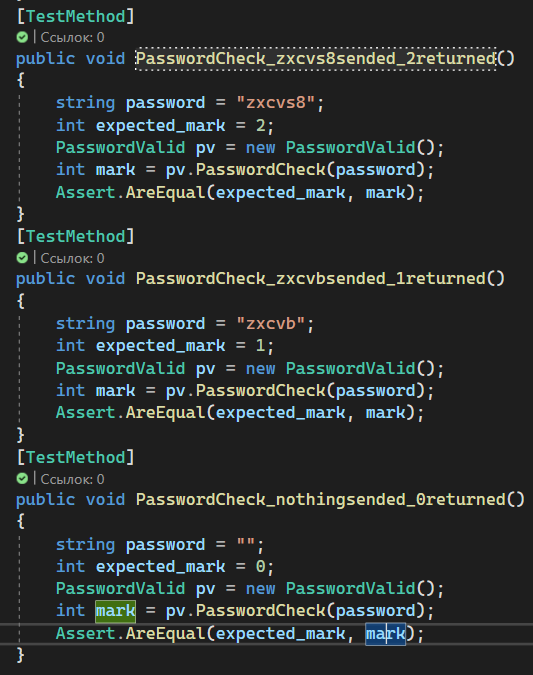


Рисунок 5 – Вторые 3 метода тестов

В тестах есть ожидаемые результаты, отправляемые данные, и фактический результат. Теперь проверим работоспособность тестов и тестируемой функции.

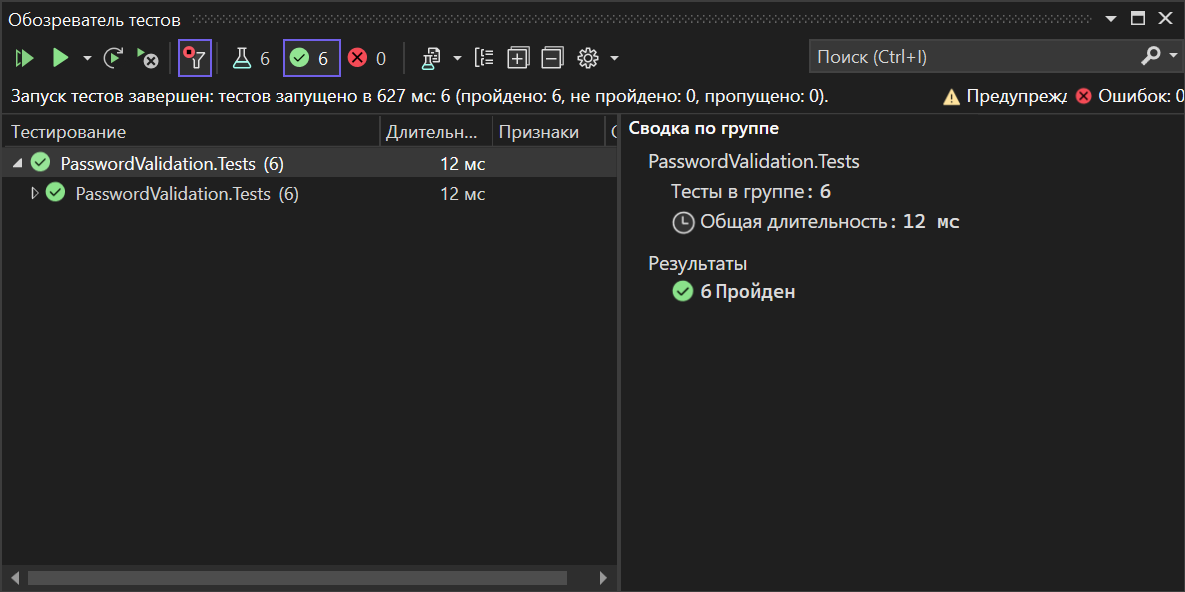


Рисунок 6 – Результат тестов

Как видим, тесты прошли успешно.

Вывод: ознакомились с unit-тестами на языке C#, научились их писать, выполнили для примера метод проверки пароля и unit-тесты для него.

ПРАКТИЧЕСКАЯ РАБОТА №2

Тема: «Test Driver Development»

Цель работы: Методом Test Driver Development протестировать некоторые функции, связанные с типом данных string.

Следуя принципу TDD, необходимо вначале создать тесты для будущих методов.

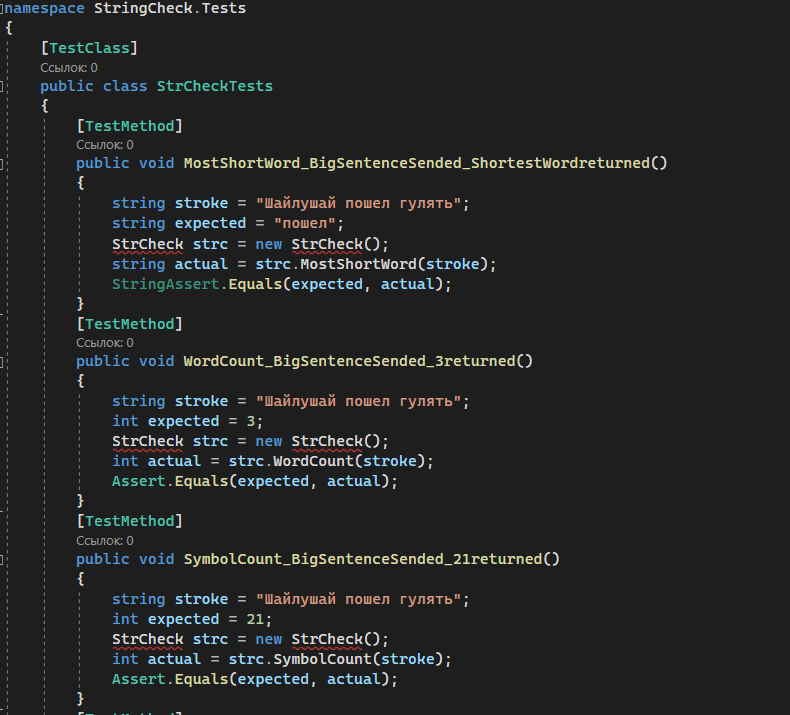


Рисунок 7 – Первые 3 теста

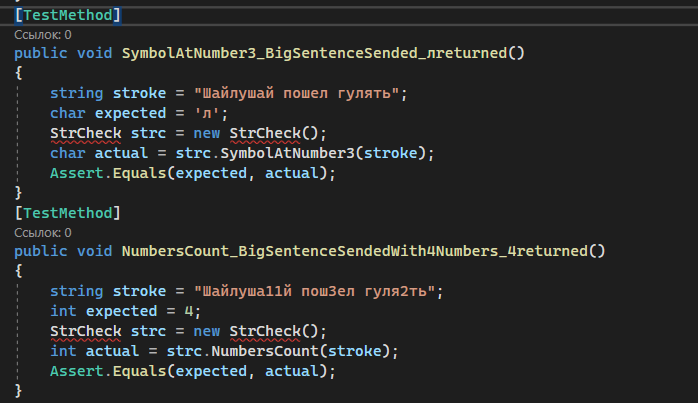


Рисунок 8 – Следующие 2 теста

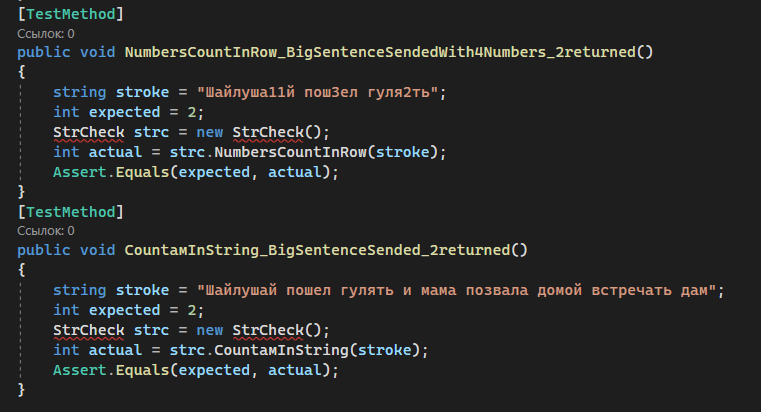


Рисунок 9 – Последние 2 теста

Теперь, нужно создать новый проект и создать внутри класс с нужными методами.

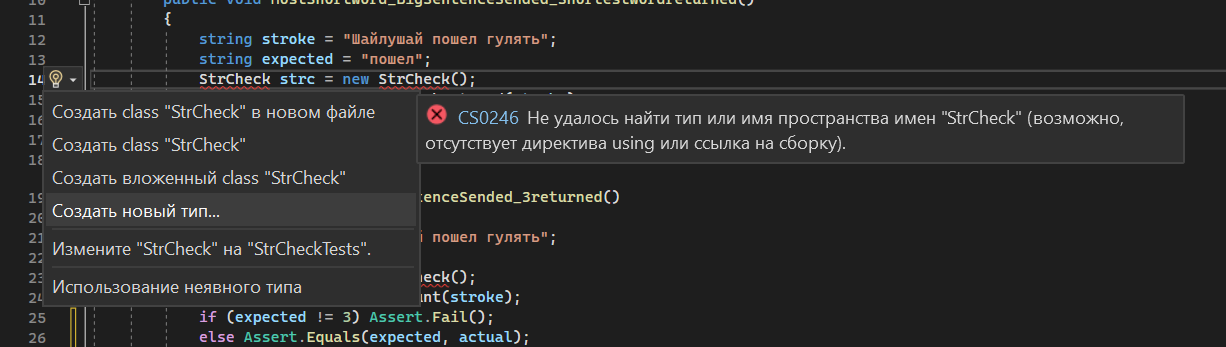


Рисунок 10 – Создание класса

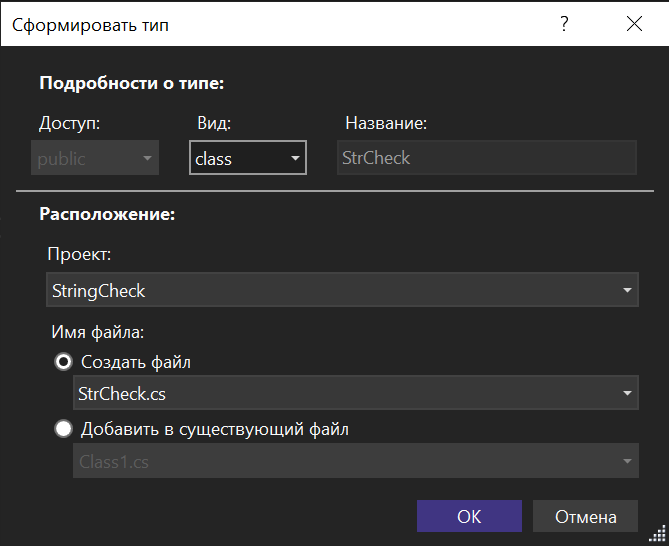


Рисунок 11 – Настройка класса

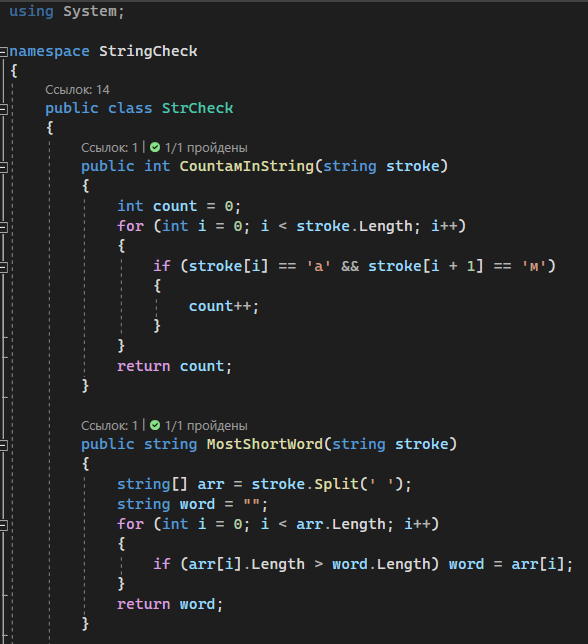


Рисунок 12 – Методы содержания строки и самого короткого слова

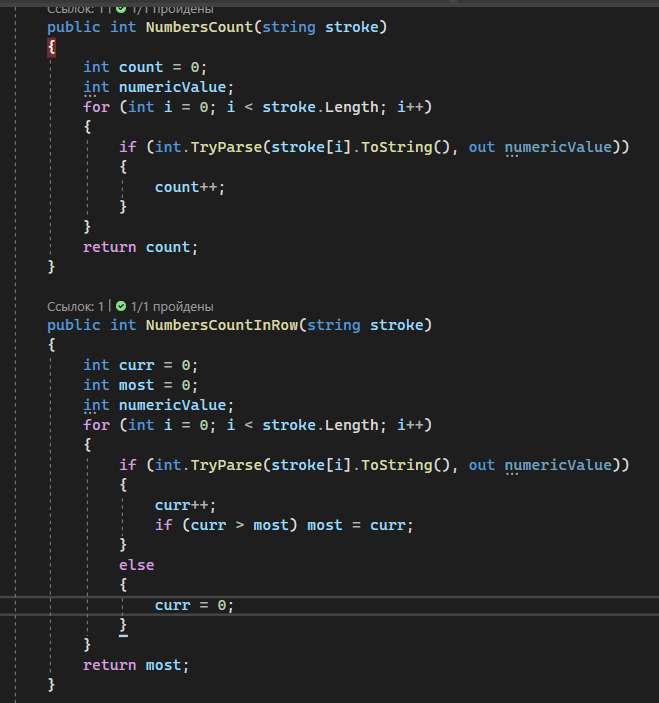


Рисунок 13 – Методы кол-ва цифр и цифр подряд

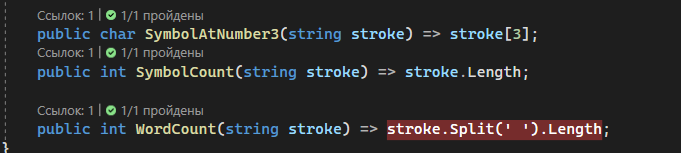


Рисунок 14 – Методы символов и кол-ва слов

Результаты работы:

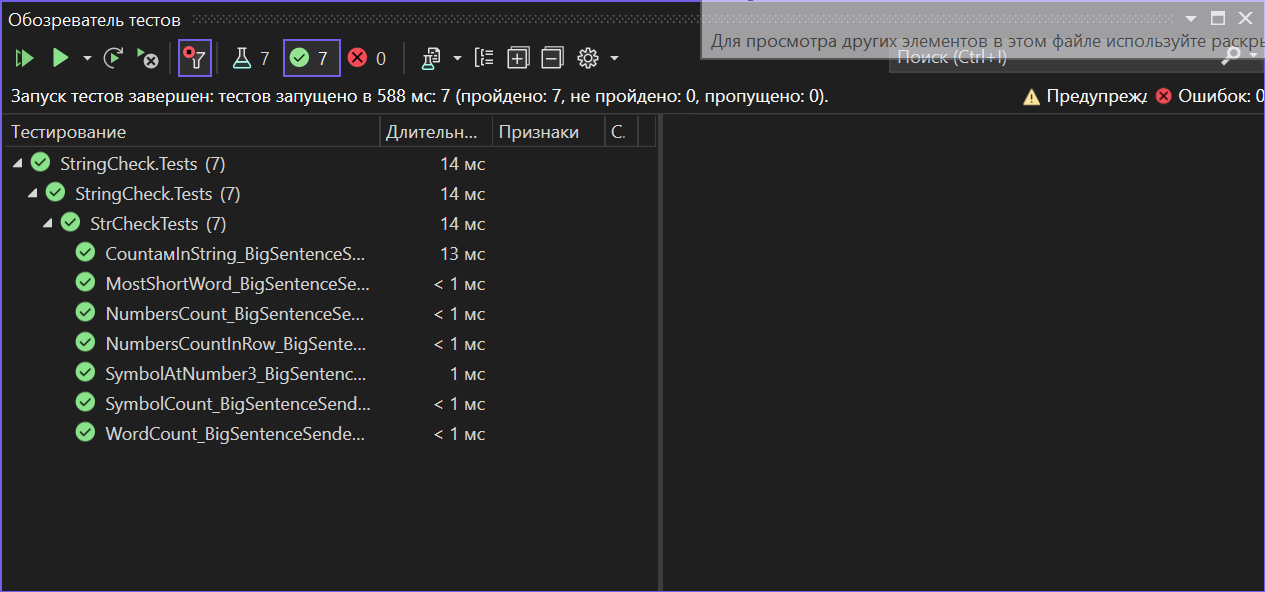


Рисунок 15 – Результат работы тестов

Вывод: Методом Test Driver Development протестированы некоторые функции, связанные с типом данных string.

ПРАКТИЧЕСКАЯ РАБОТА №3

Тема: «CollectionAssert»

Цель работы: научиться работать с методами класса CollectionAssert, создав две функции логики и четыре теста для них.

Начать работу следует с создания методов логики. Всего их будет два: один для нахождения дискриминанта из 3-х чисел, другой для нахождения процента от числа.

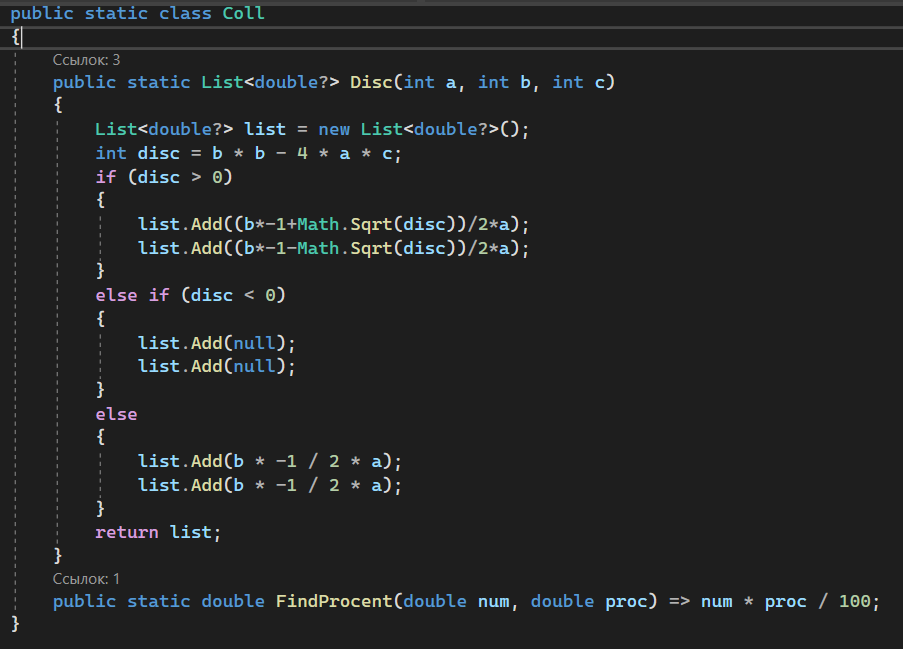


Рисунок 16 – Методы нахождения дискриминанта и процента

После создания логики необходимо создать тесты. Тестов в общей сумме будет 4, три – для каждого вида дискриминанта, и один – для нахождения процента от числа. В классе тестов также можно использовать инициализатор для инициализации каких-либо переменных.

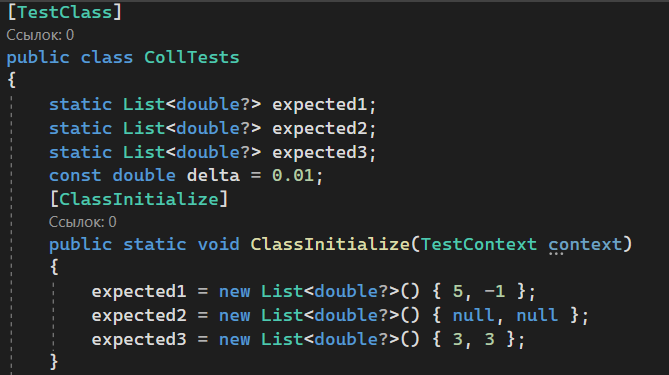


Рисунок 17 – Инициализация класса тестов

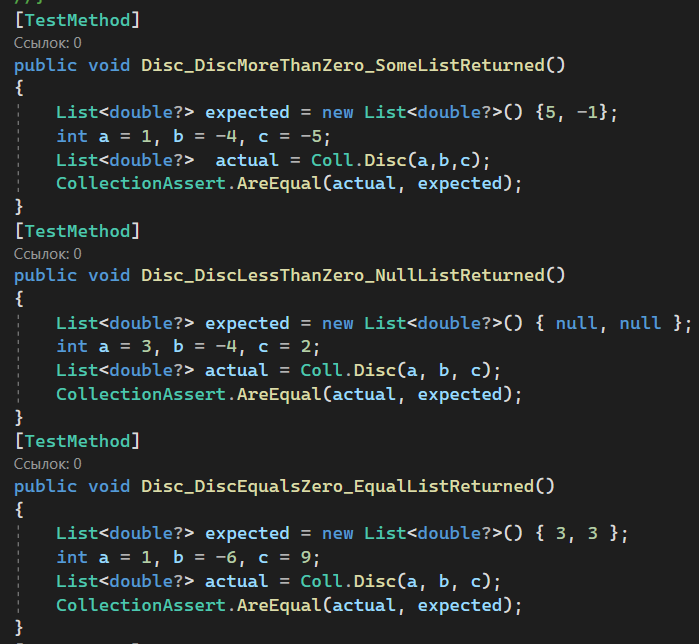


Рисунок 18 – Тесты для метода нахождения дискриминанта

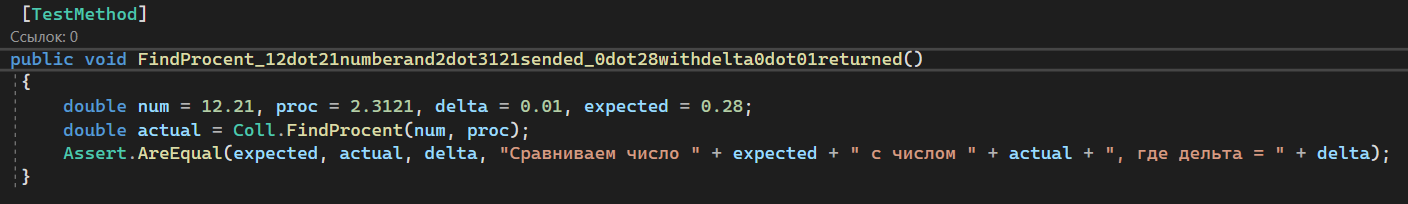


Рисунок 19 – Тест для метола нахождения процента от числа

Структура проекта выглядит следующим образом:

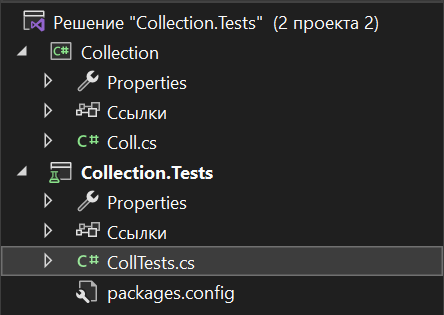


Рисунок 20 – Структура проекта

Результат работы:

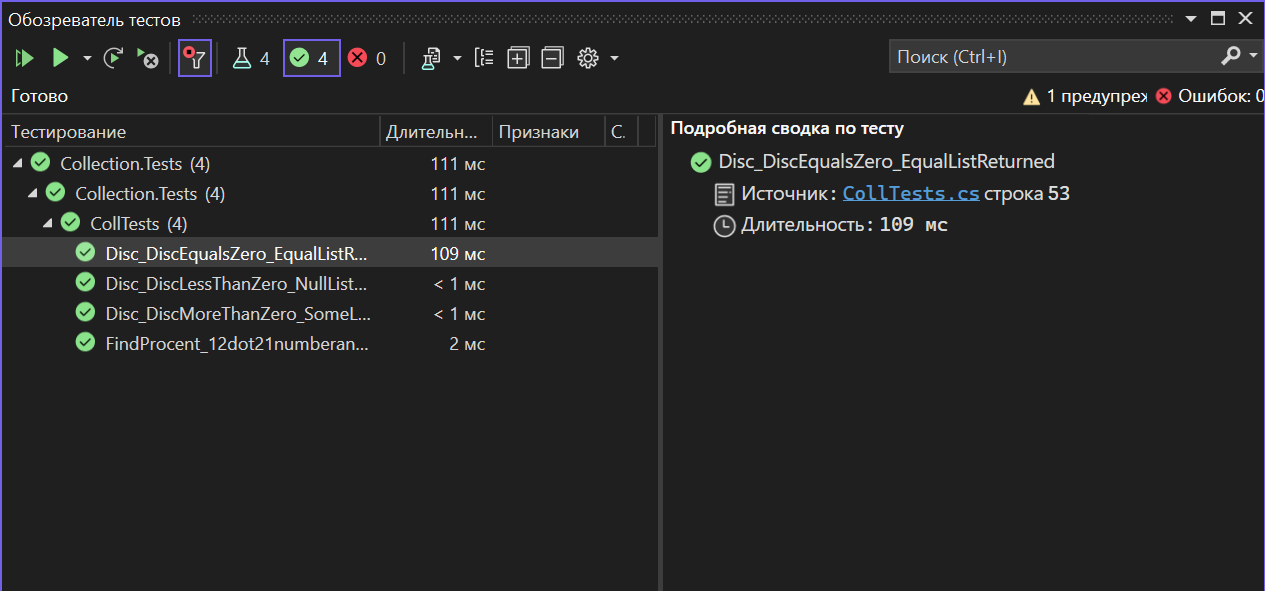


Рисунок 21 – Результат работы

Вывод: Были созданы две функции логики и четыре теста для них, попутно научившись работать с методами класса CollectionAssert.

ПРАКТИЧЕСКАЯ РАБОТА №4

Тема: «Stub-тест»

Цель работы: Необходимо создать консольное приложение с интеграционной связью, создать в этом приложении заглушку test Stub и протестировать Unit тестом.

Тема приложения: Проверка наличия в папке файлов с определенным расширением (Пример: .txt).

Начать следует с создания начальной точки программы.

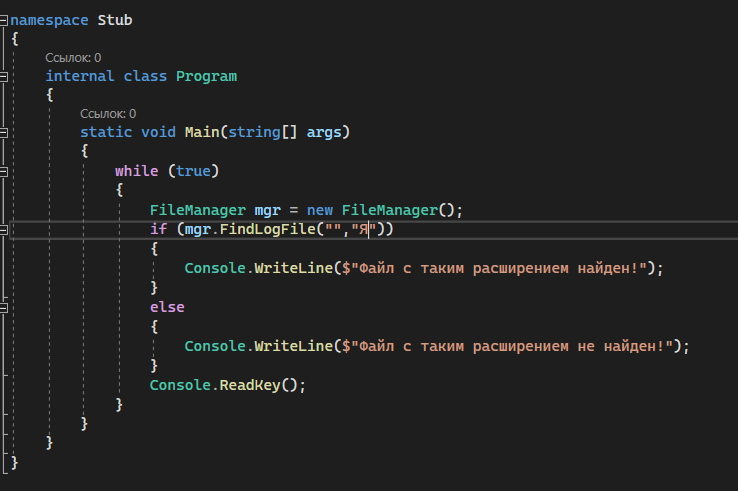


Рисунок 22 – Начало программы

В FileManager есть две развилки. Одна – рабочий метод поиска файлов, другая – заглушка, имитирующая возврат файлов.

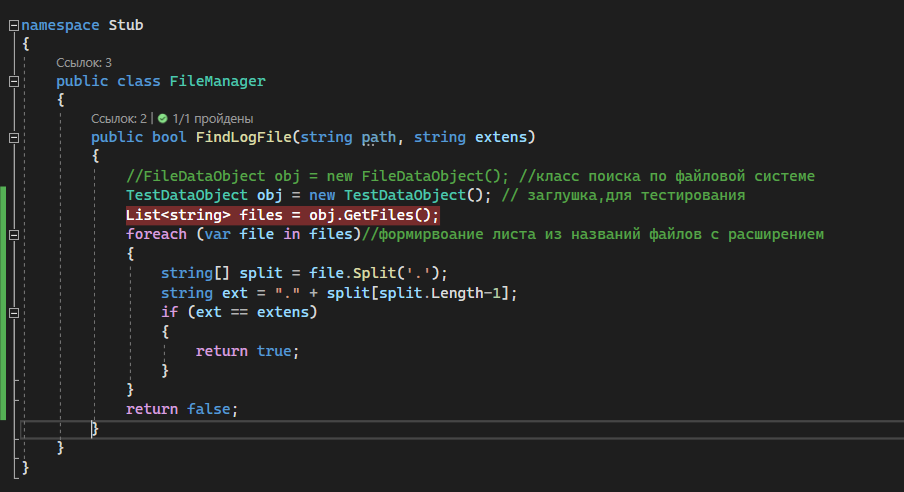


Рисунок 23 – Получение списка файлов и проверка их расширения

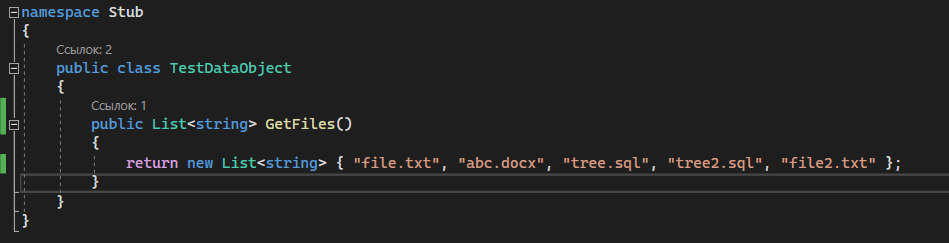


Рисунок 24 – Работа заглушки

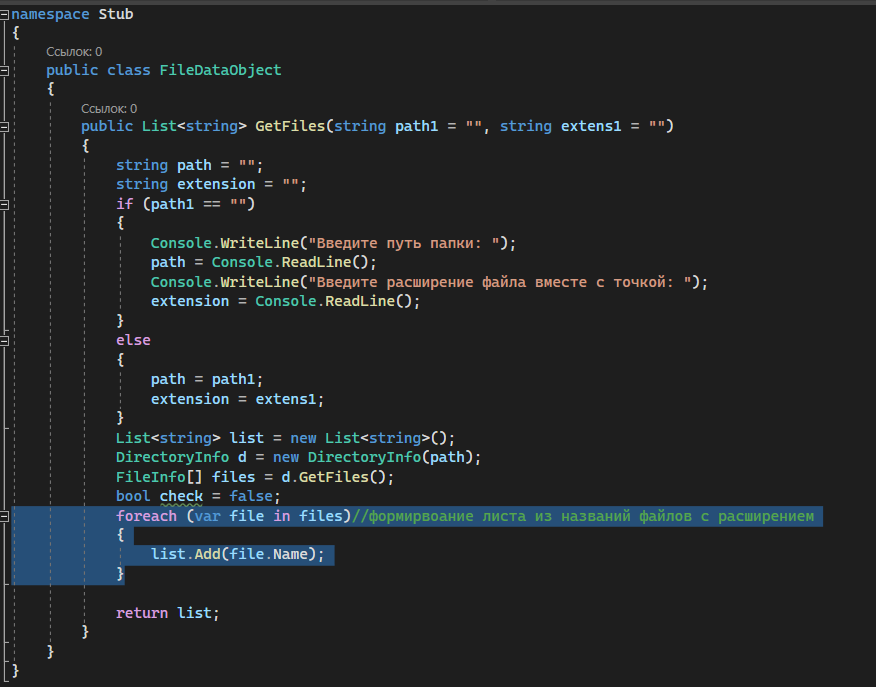


Рисунок 25 – Работа нахождения файлов

Тест, в котором вызываются последующие методы выглядит следующим образом:

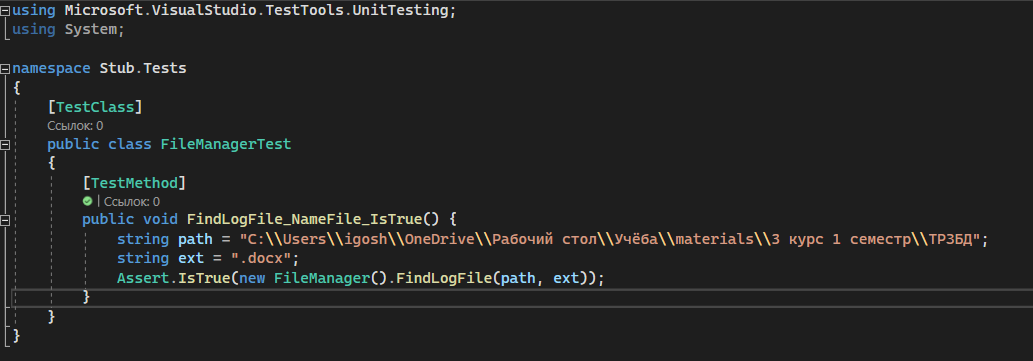


Рисунок 26 – Сам тест

Результаты работы:

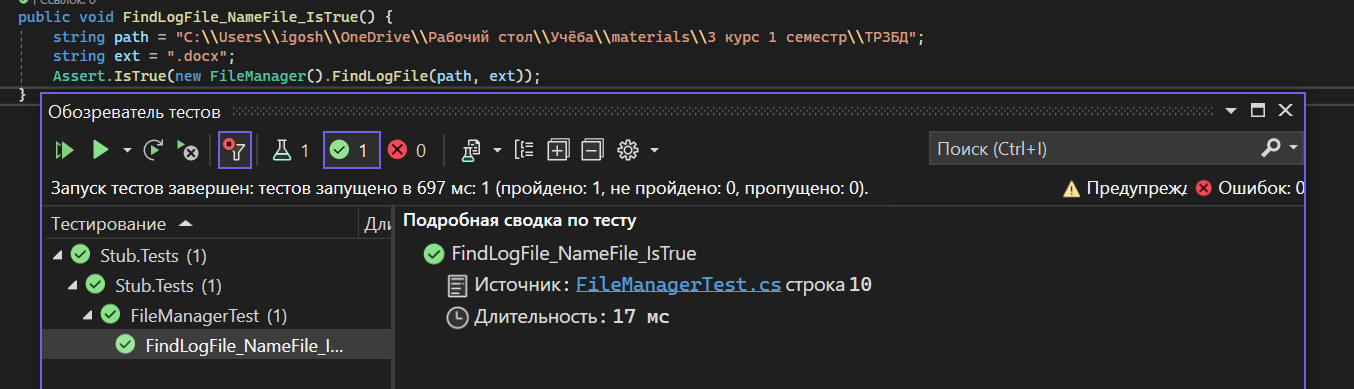


Рисунок 27 – Вызов теста с расширением .docx

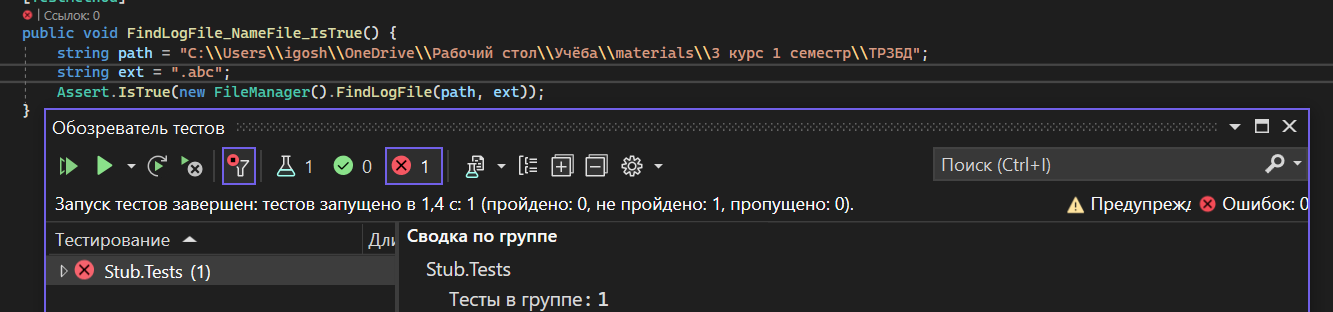


Рисунок 28 – Вызов теста с расширением .abc

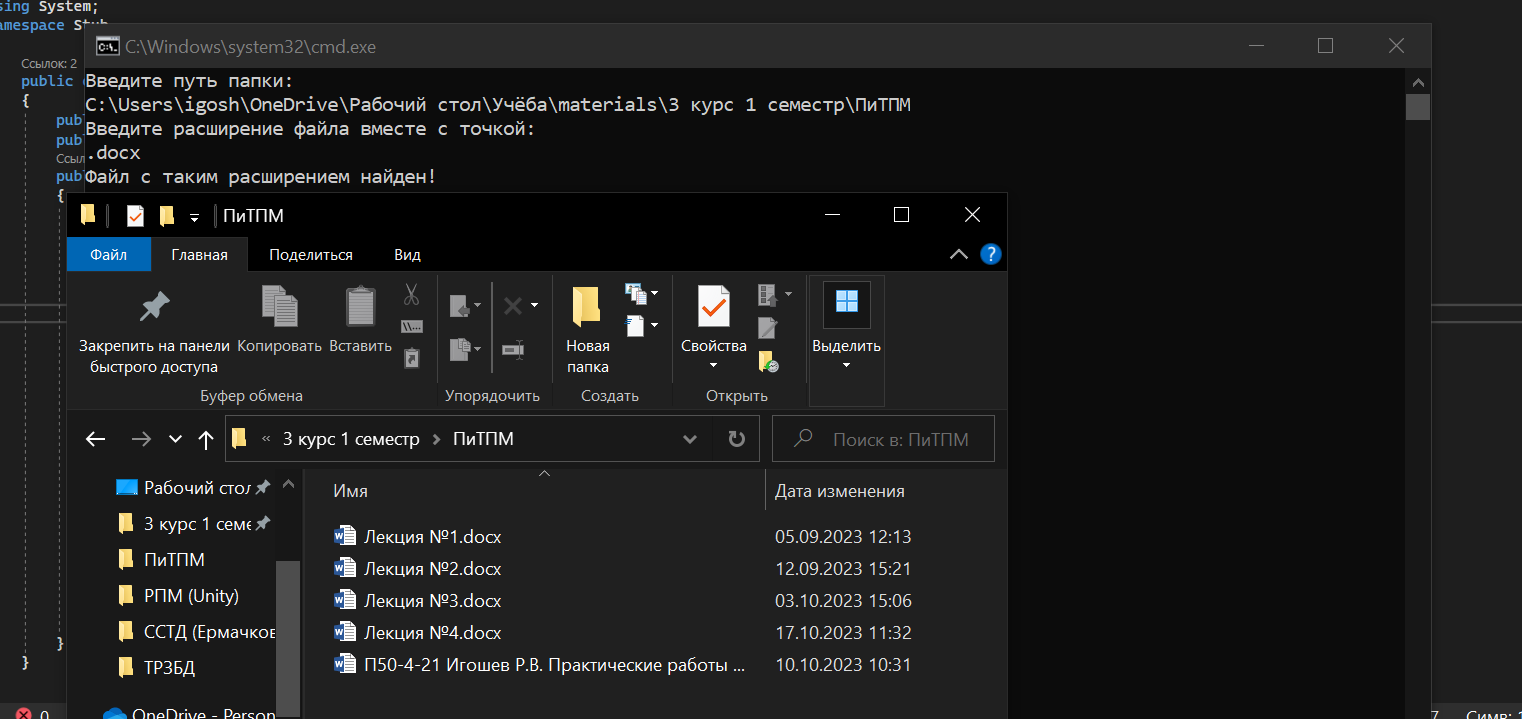


Рисунок 29 – Работа программы без заглушки

Вывод: В процессе работы было создано консольное с интеграционной связью, была создана в этом приложении заглушка test Stub и протестирована Unit тестом.

ПРАКТИЧЕСКАЯ РАБОТА №5

Тема: «JUnit5»

Цель работы: Необходимо создать программу Java. При тестировании использовать JUnit5. Использовать аннотации из прикрепленного материала. Весь прикрепленный материал изучить и выучить.

Для начала необходимо реализовать файл с логикой программы.

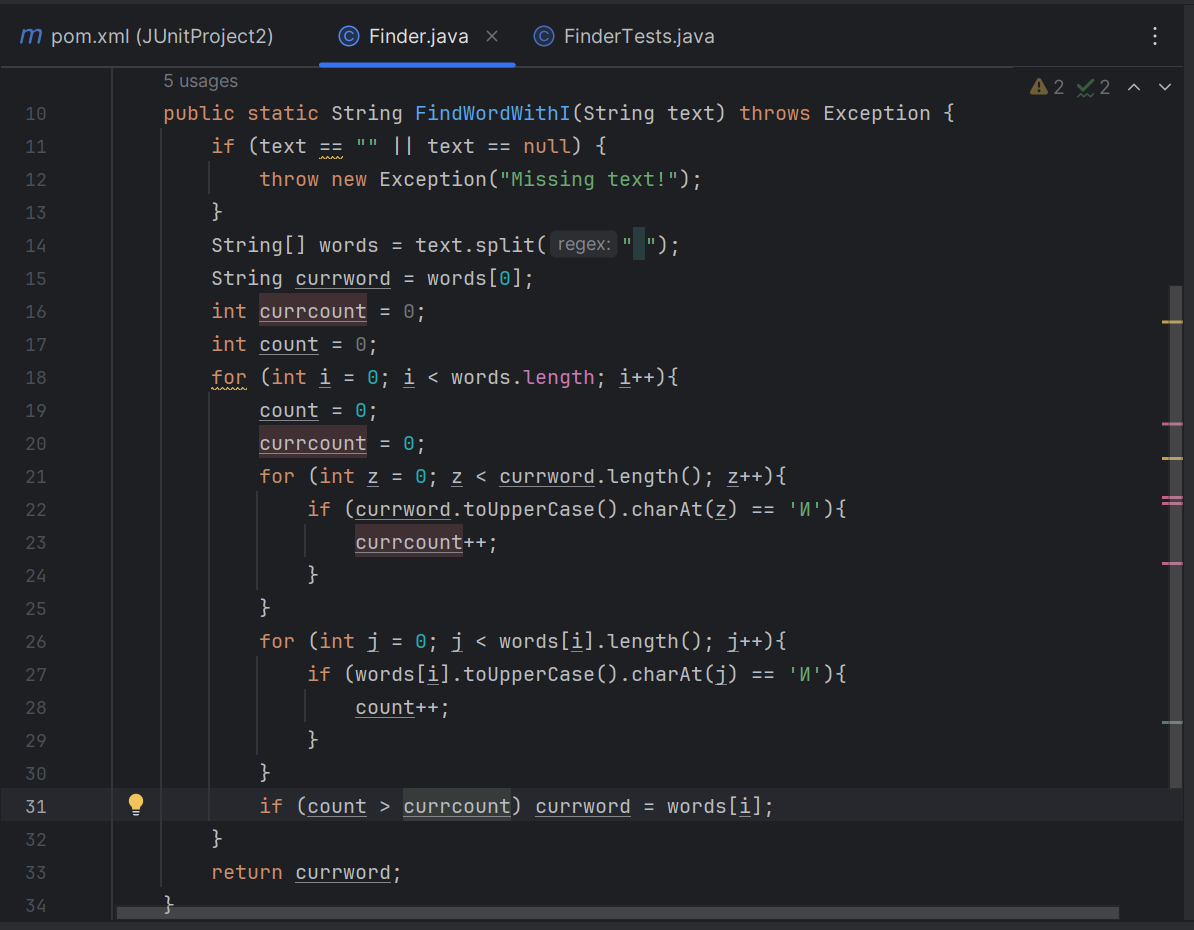


Рисунок 30 – Метод нахождения слова с наибольшим кол-вом букв «и»

Теперь, к созданной логике необходимо реализовать тесты.

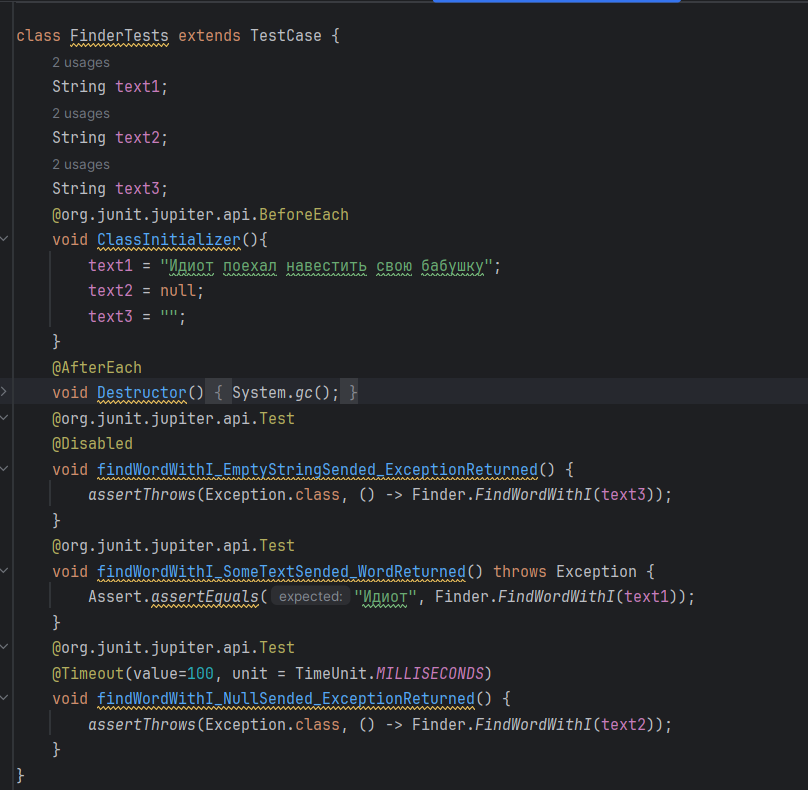


Рисунок 31 – Тесты

Результат работы:

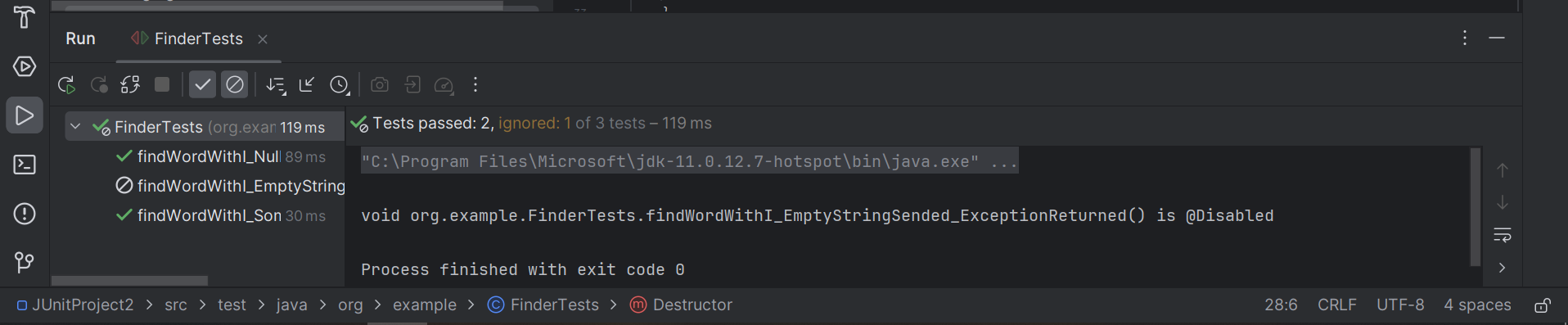


Рисунок 32 – Результат работы тестов

Вывод: Необходимо создать программу Java. При тестировании использовать JUnit5. Использовать аннотации из прикрепленного материала. Весь прикрепленный материал изучить и выучить.