**Лабораторная работа №2**

**Использование заглушек для разрыва зависимостей**

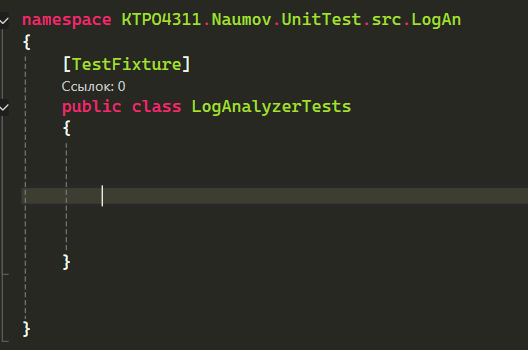
**Цель работы:**

Приобретение практических навыков использования заглушен для разрыва зависимостей при автономном тестировании модулей, практика использования тестового каркаса NUnit.

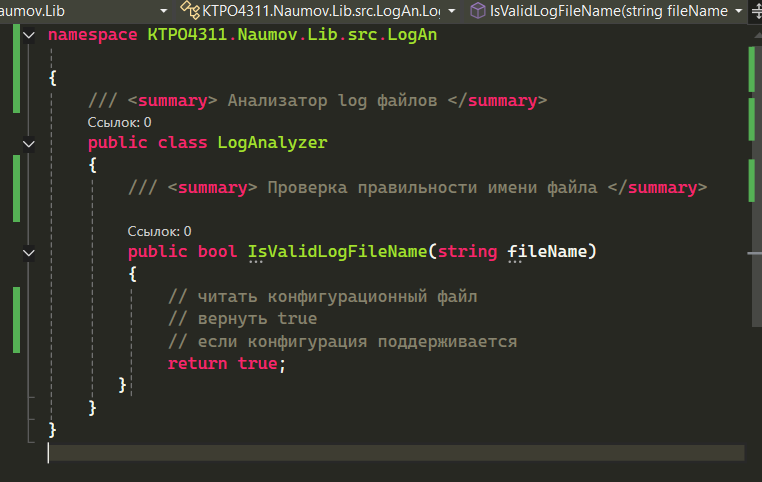
**Порядок выполнения работы**

**Подготовка проекта**

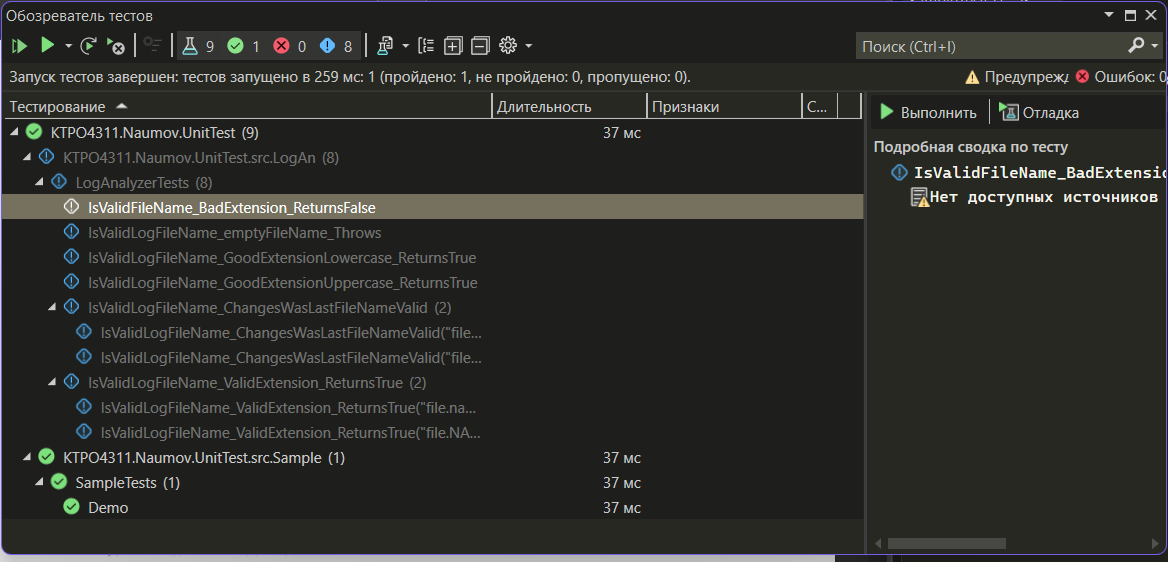
1. Удаляем тесты из класса LogAnalyzerTests.



1. Приводим класс LogAnalyzer к следующему виду.

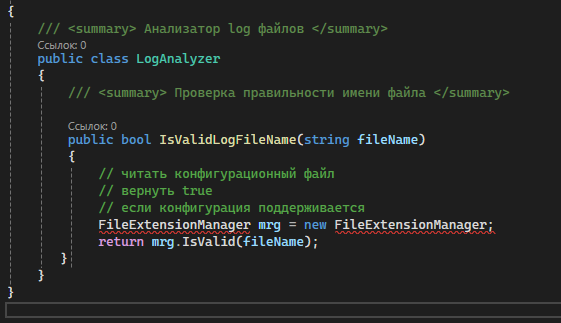


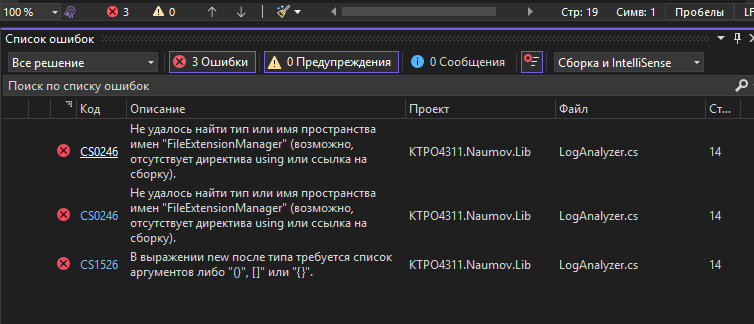
1. Выполняем тесты.



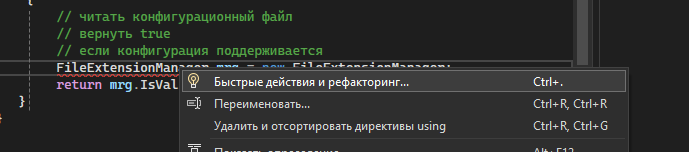
**Рефакторинг проекта для повышения тестопригодности: выделение в отдельный класс операций с файловой системой**

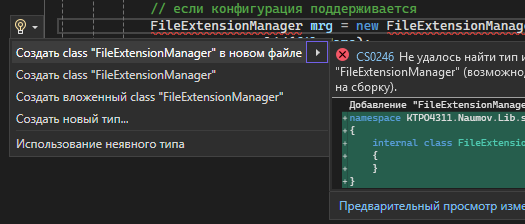
1.

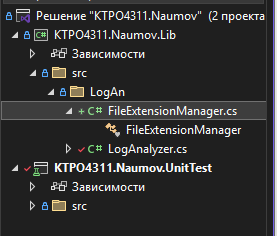
****

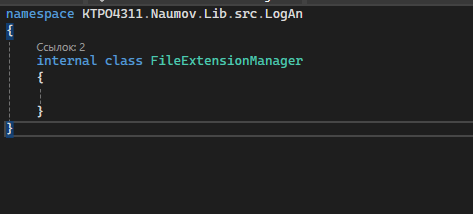
****

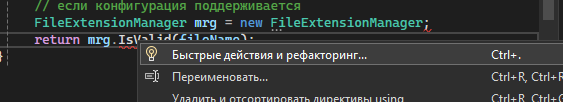
2.

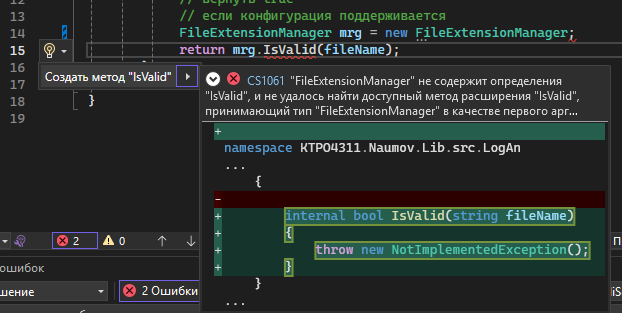
****

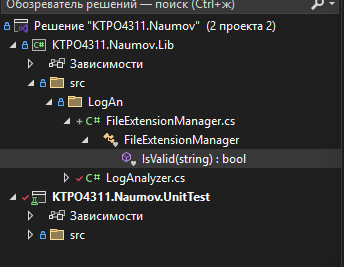
****

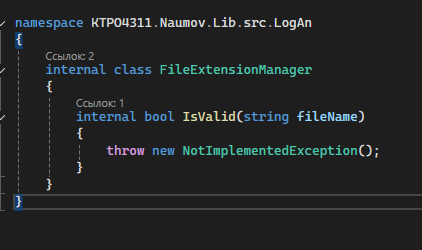
****

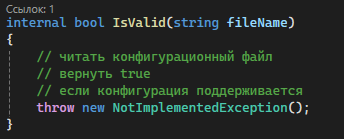
****

****

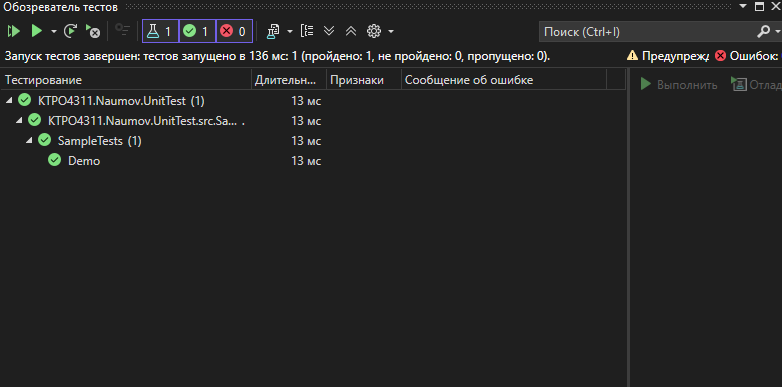
****

****

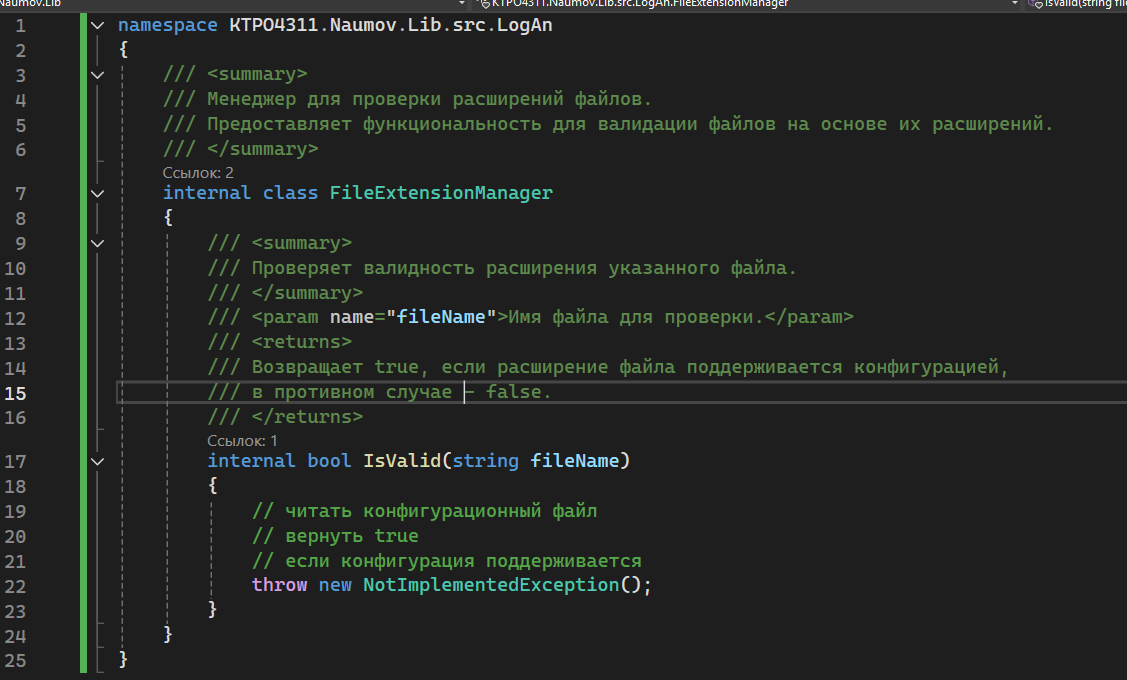
****

****

Запускаем тесты

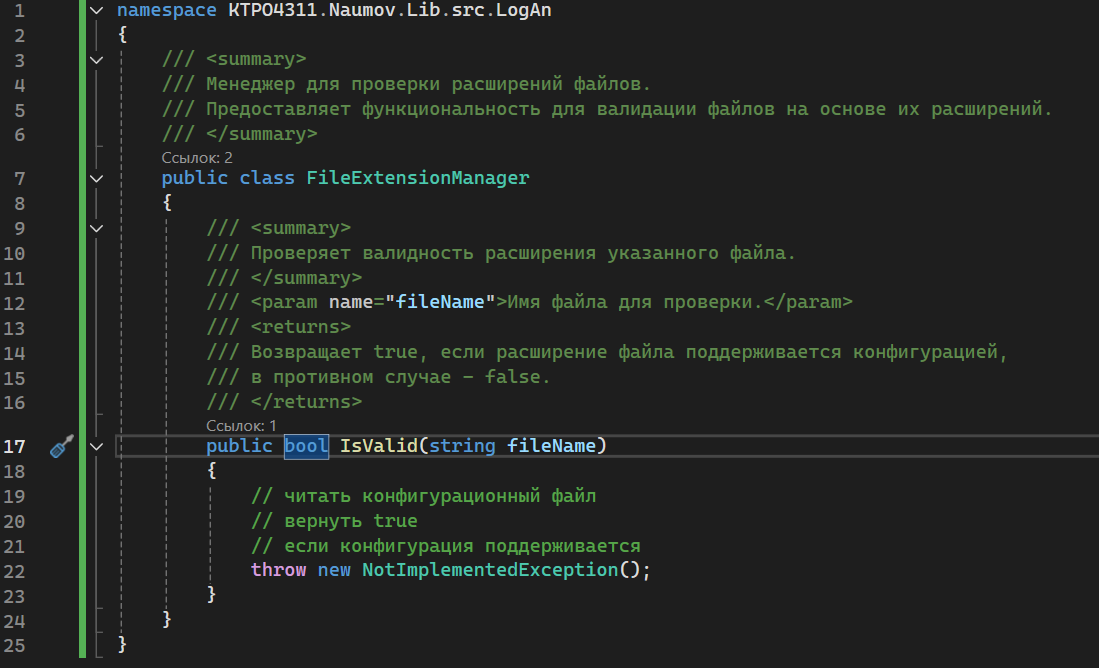
****

Добавляем summary для класса и метода

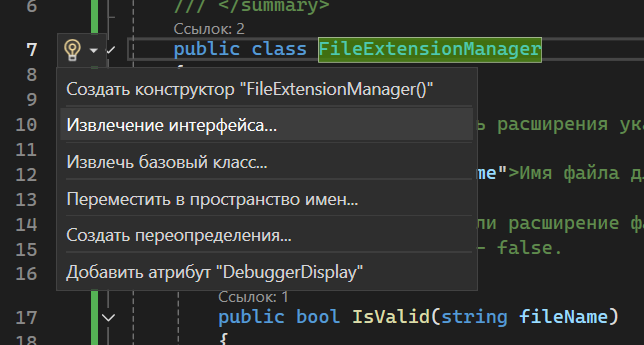
**я**

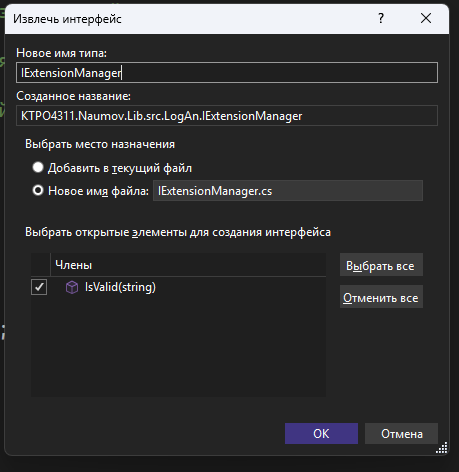
**Рефакторинг проекта для повышения тестопригодности: выделение интерфейса**

1. Заменяем модификатор доступа

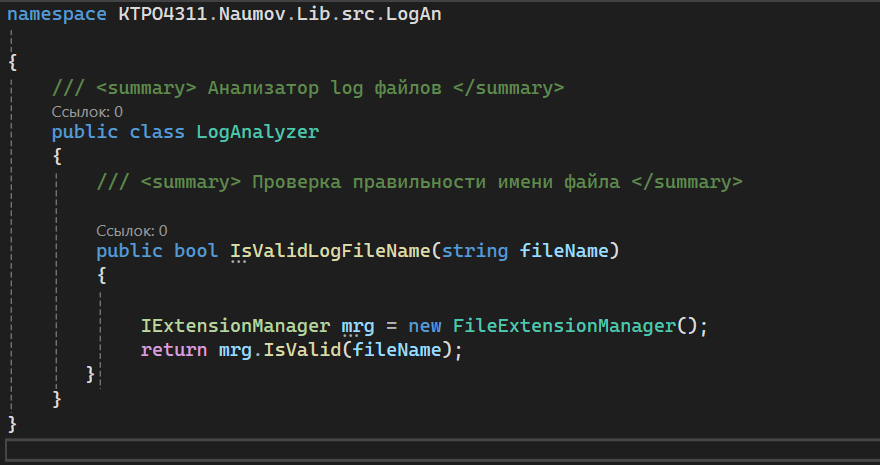


1. Выделяем интерфейс FileExtensionManager



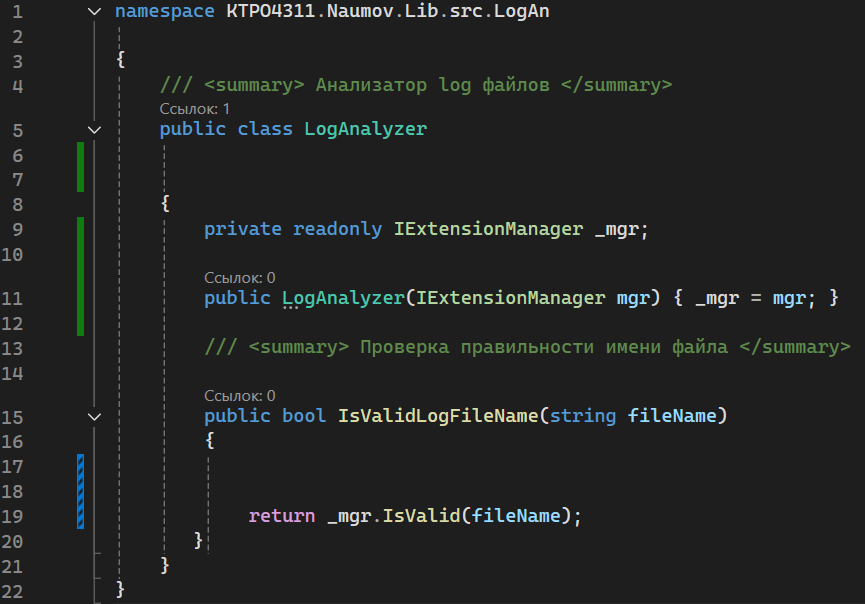


1. Объявляем переменную диспетчера расширений как интерфейс

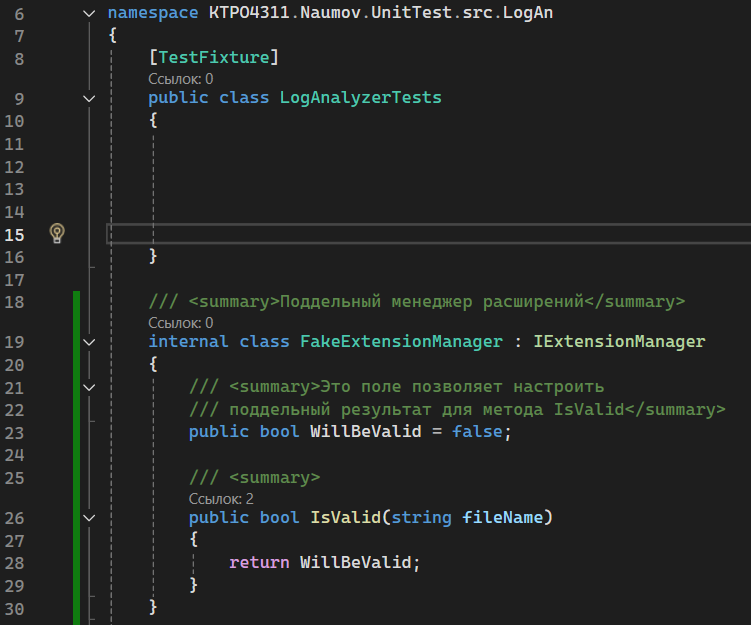


**Внедрение зависимости: Внедрение подделки через конструктор**

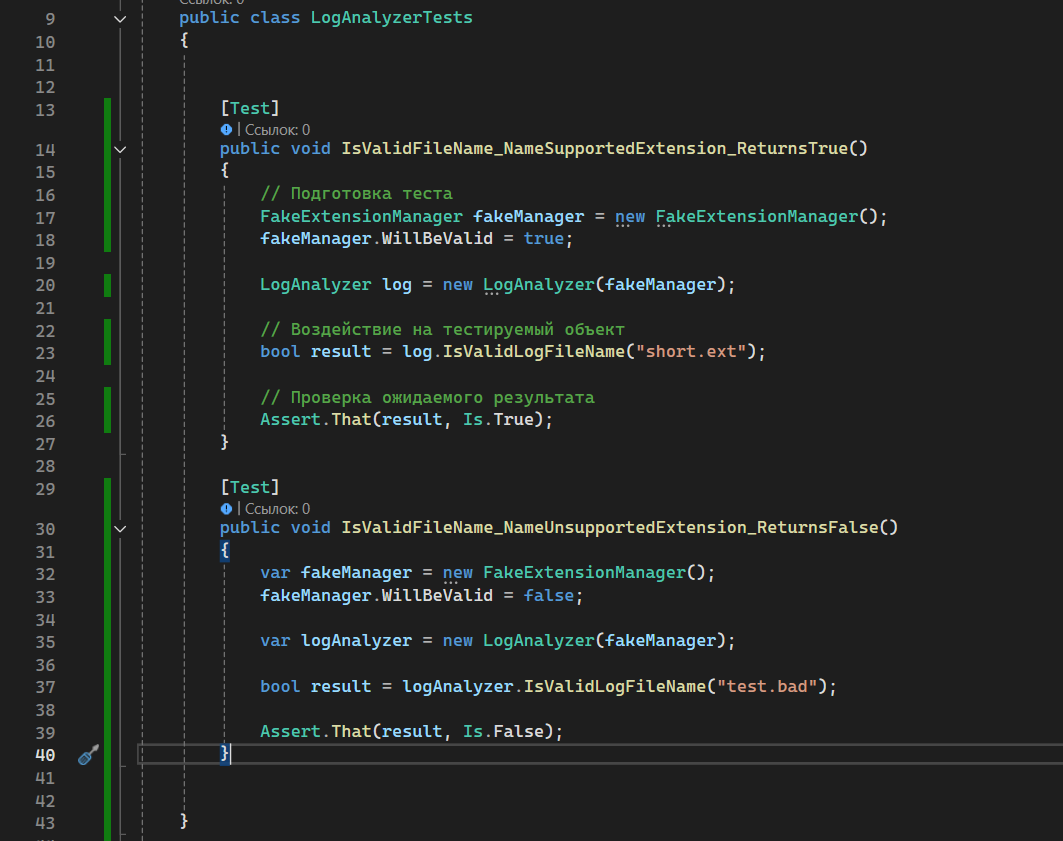
1. Создаём поле типа IExtensionManager и конструктор. Заменяем создание экземпляра класса на обращение к полю.



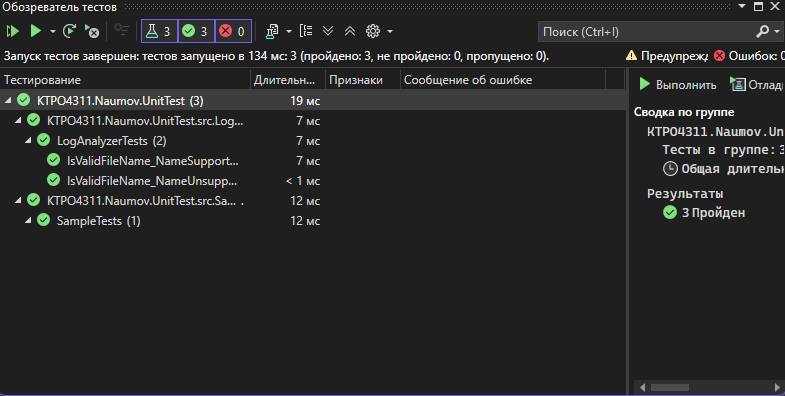
1. Создаём заглушку FakeExtensionManager



1. Создаём тестовые методы

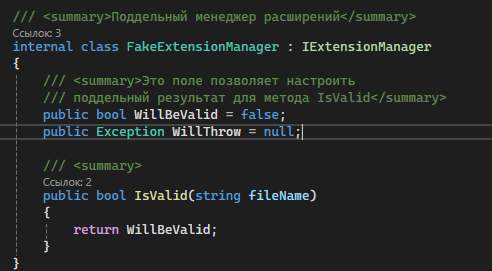


1. Прогоняем тесты

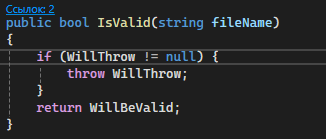


**Имитация возникновения исключения во внешнем классе**

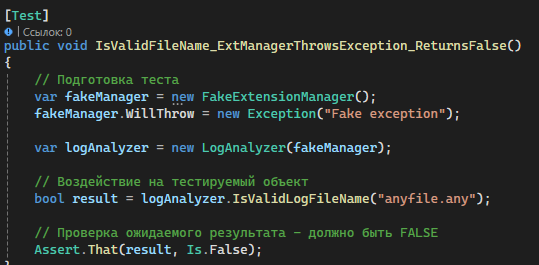
1. Добавляем поле WillThrow в класс FakeExtensionManager



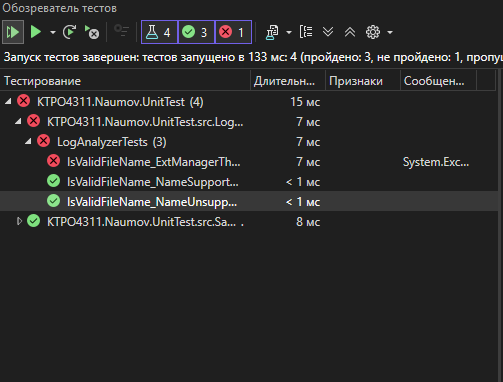
1. Дорабатываем метод IsValid



1. Создаём тестовый метод для сценария “Если диспетчер расширений вызвал исключение, вернуть false”.

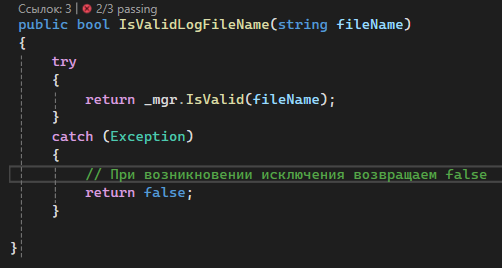


1. Выполним тесты

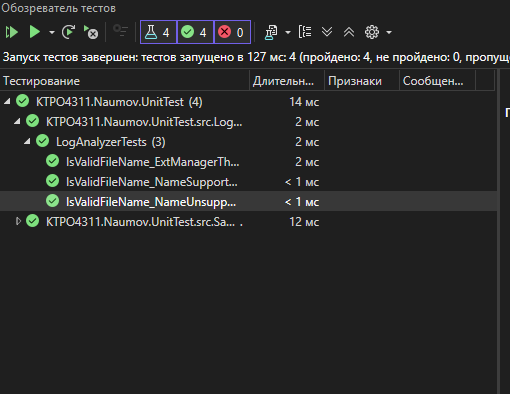


Тест не прошел, поскольку метод IsValidLogFileName ещё не обрабатывает исключения.

1. Добавим обработку исключения.

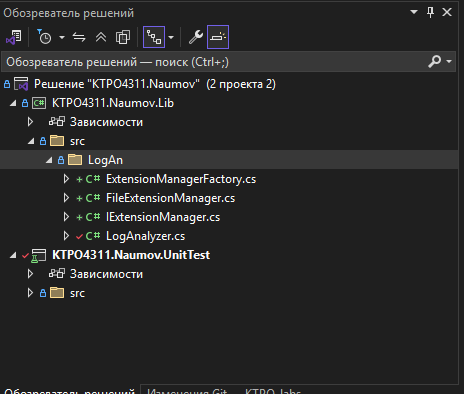


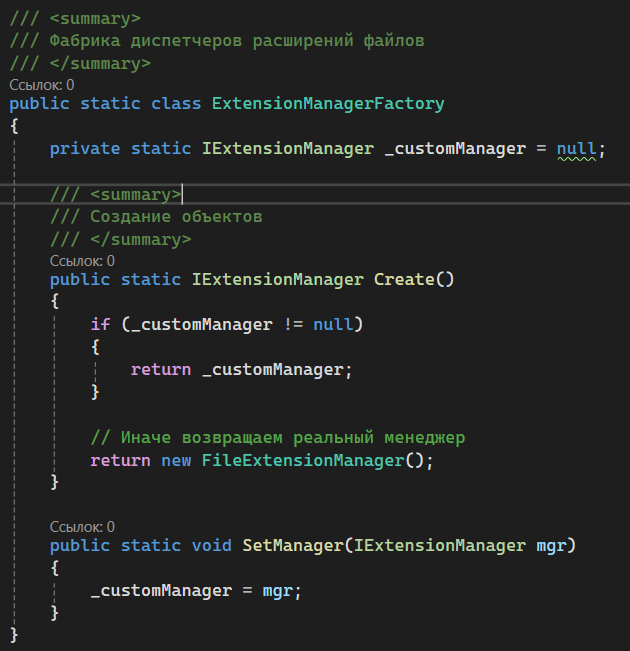
1. Выполним тесты.



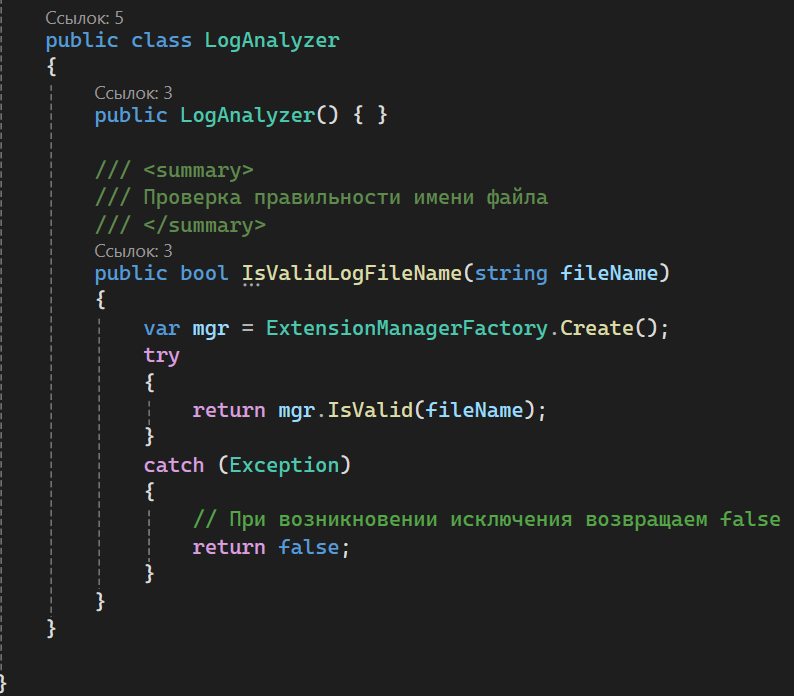
**Внедрение зависимости. Внедрение подделки через фабрику.**

1. Создаём резервную копию проекта.
2. Создаём класс ExtensionManagerFactory

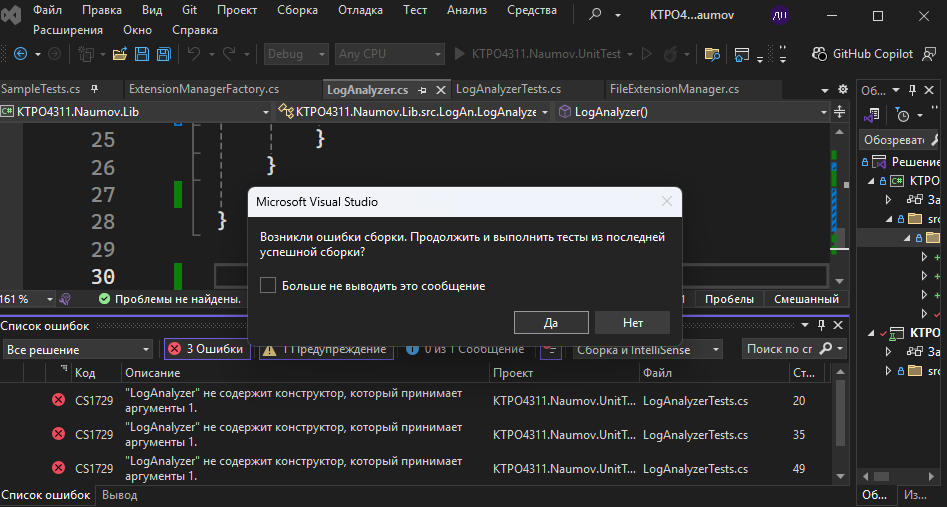




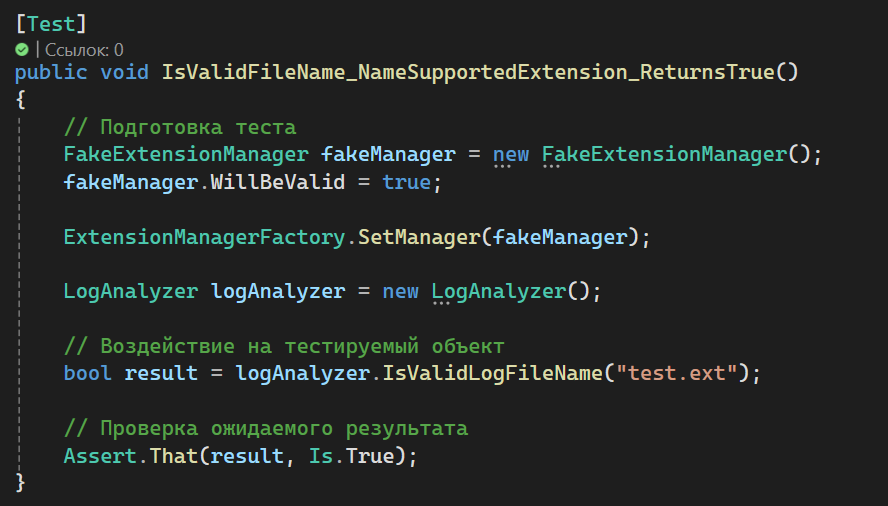
1. Изменяем LogAnalyzer

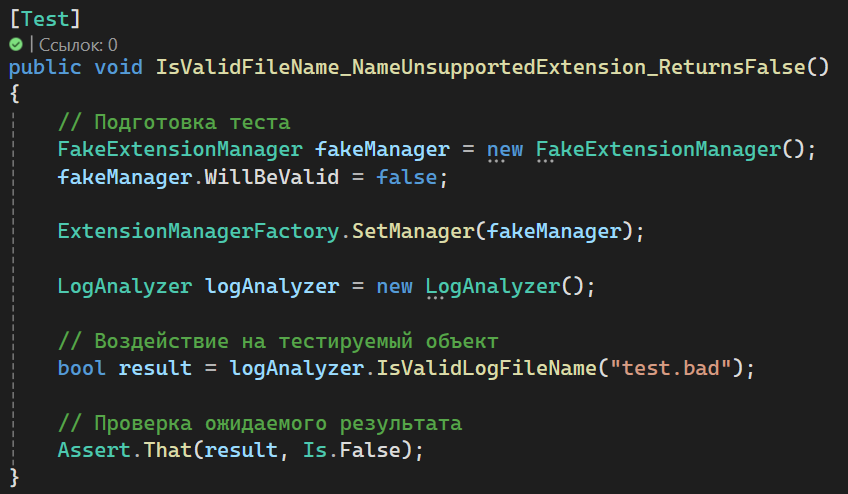


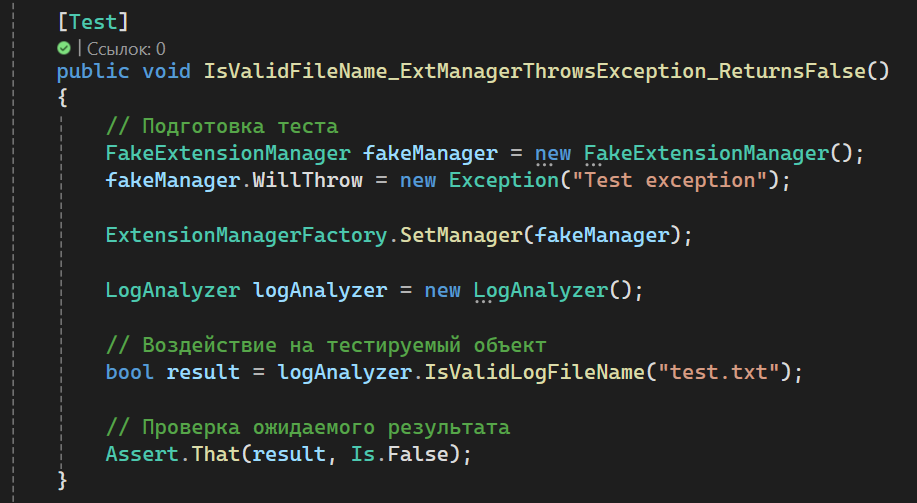
Запускаем тесты



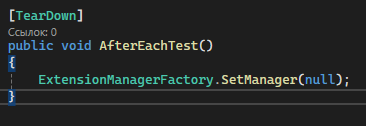
1. Исправляем тесты. Добавляем конфигурирование фабричного класса поддельным диспетчером расширений с помощью метода SetManager.



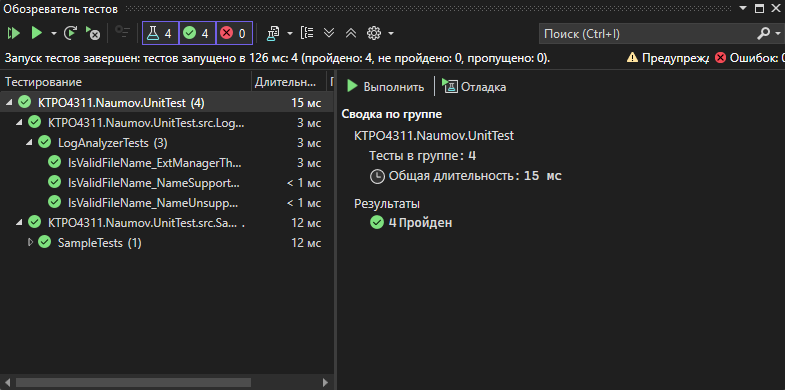




1. Добавляем метод AfterEachTest в тестовый класс.

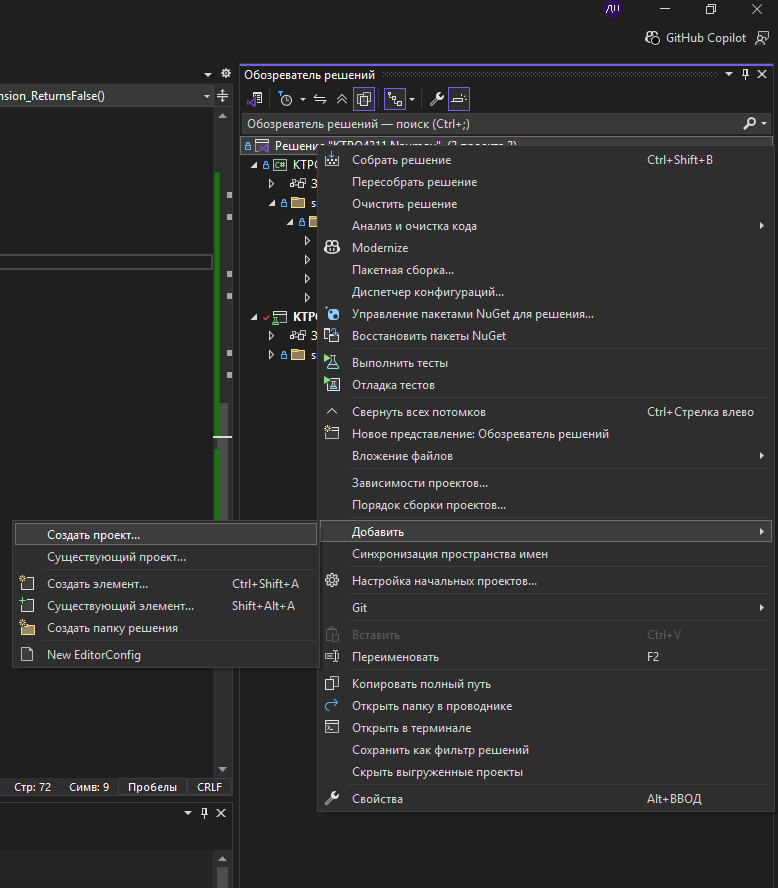


1. Выполним тесты.

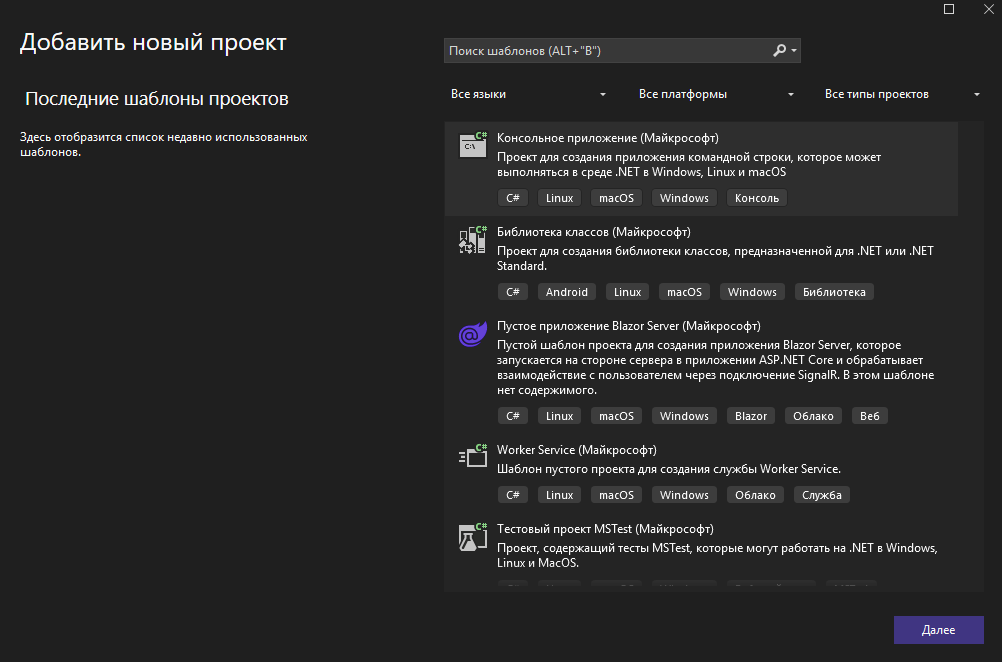


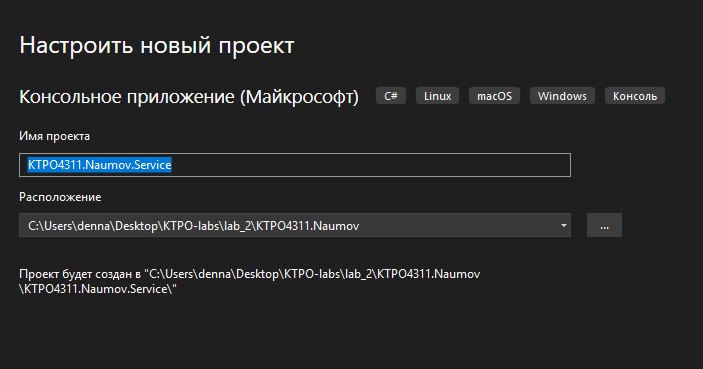
**Реализовать диспетчер расширений и клиентское приложение.**

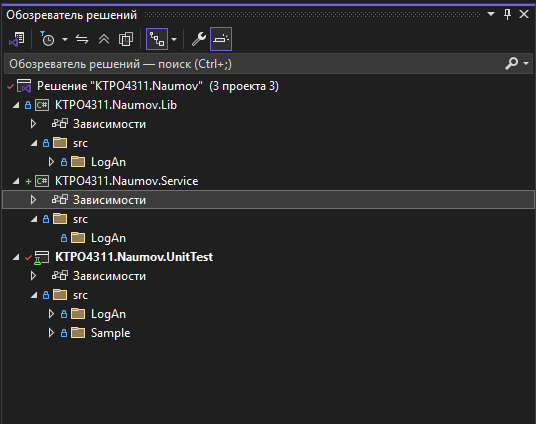
1. Добавляем в решение новый проект



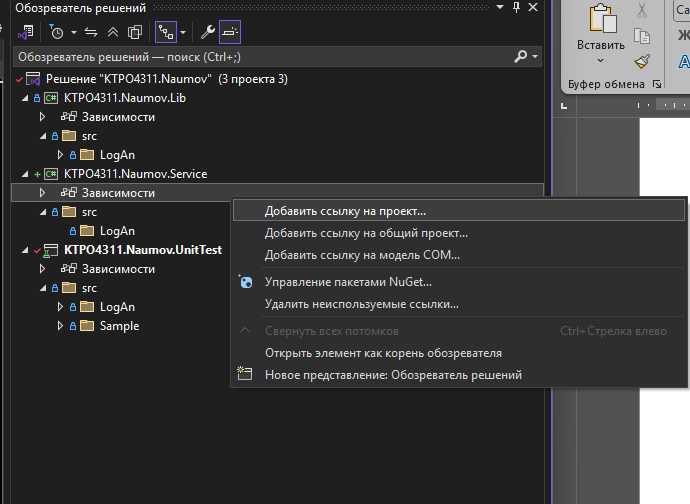
1. **ПКМ**





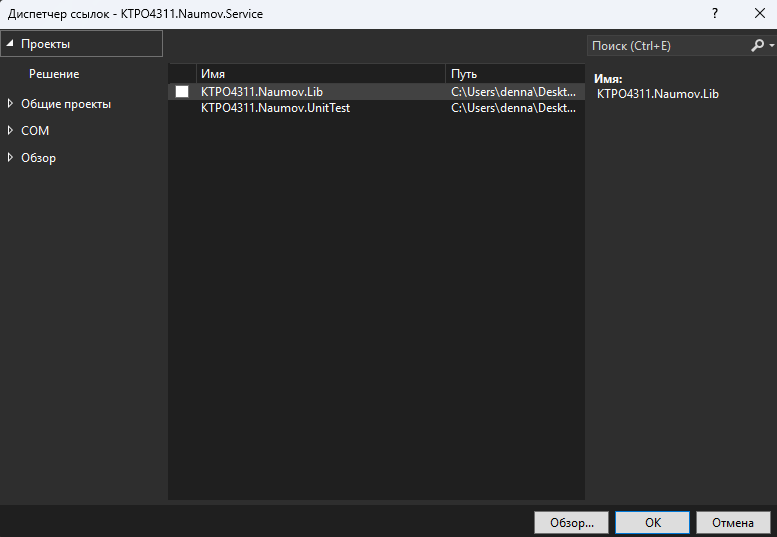


Добавляем ссылку на проект Lib



**2.**

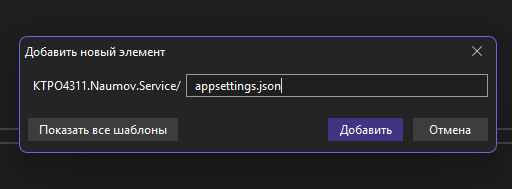
1. **ПКМ**

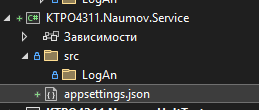


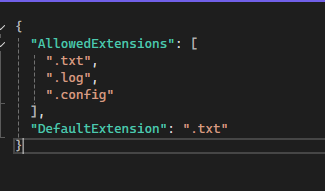
**2.**

**1.**

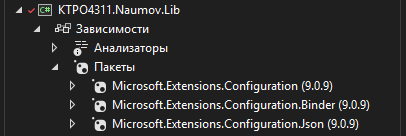
1. Добавляем файл настроек в корень проекта.



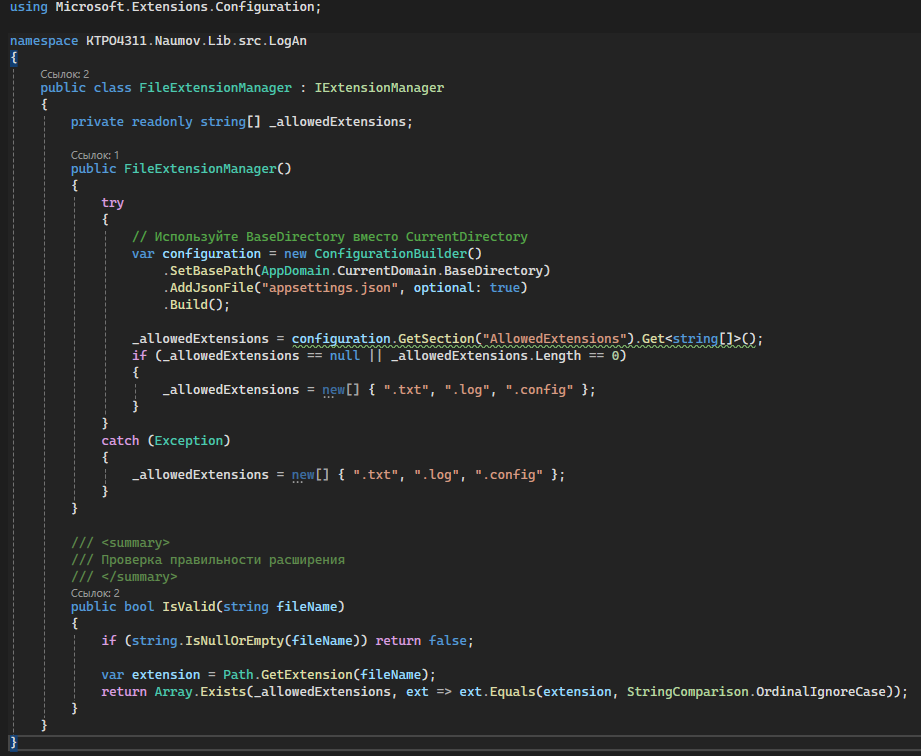




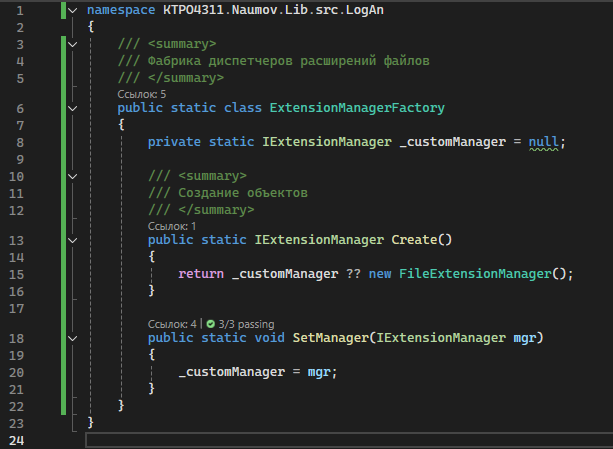
Устанавливаем необходимые пакеты.



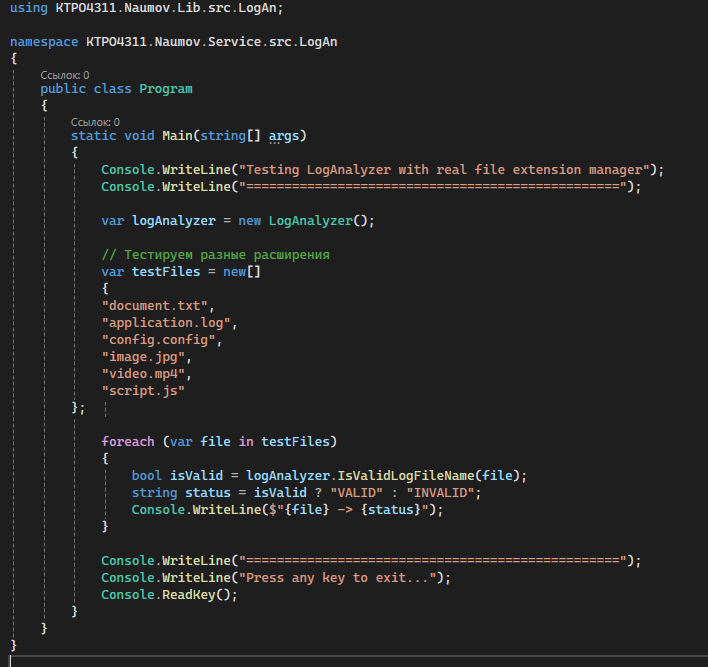
1. Обновляем FileExtensionManager



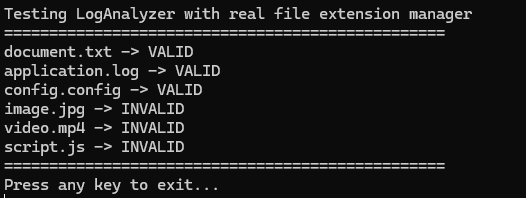
Обновляем ExtensionManagerFactory.



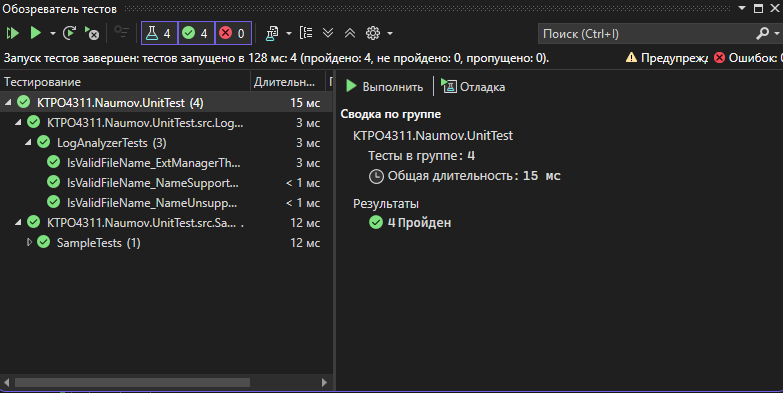
1. В консольном приложении тестируем LogAnalyzer.



1. Выполняем сборку и запускаем проект Service.



1. Выполним тесты.



**Выводы**

В ходе выполнения лабораторной работы были успешно приобретены практические навыки использования заглушек для разрыва зависимостей при автономном тестировании модулей. Основным достижением стало освоение техники создания FakeExtensionManager, который позволил полностью изолировать тестируемый код от файловой системы. Был реализован механизм настройки поведения заглушки через свойства WillBeValid и WillThrow, что обеспечило возможность тестирования различных сценариев, включая исключительные ситуации.

Важным этапом работы стало применение принципов Dependency Injection через два подхода: внедрение зависимости через конструктор и через фабрику. Освоение паттерна "Factory" позволило организовать централизованное управление созданием объектов, а работа с абстракцией IExtensionManager вместо конкретных реализаций продемонстрировала преимущества слабой связанности компонентов.

Проведенный рефакторинг кода значительно повысил тестопригодность системы - был выделен отдельный класс FileExtensionManager, создан интерфейс IExtensionManager и реализован принцип инверсии зависимостей. Это позволило создать комплексную систему тестирования, покрывающую все сценарии использования: валидные и невалидные расширения, а также исключительные ситуации.