

## Лабораторная работа №2.

### Использование заглушек для разрыва зависимостей

#### Цель работы

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

#### Порядок выполнения работы

##### 1. Подготовка проекта

В решение, полученном после выполнения первой л.р. внести следующие изменения:

1. Удалите тесты из класса LogAnalyzerTests
2. Тестируемый класс LogAnalyzer приведите к виду как показано на рисунке. Псевдокод в методе LogAnalyzer.IsValidLogFileName будет имитировать наличие настоящей операции чтения.

```
namespace KТРО4310.Ivanov.Lib.src.LogAn
{
    /// <summary>Анализатор лог. файлов</summary>
    Ссылка: 0
    public class LogAnalyzer
    {
        /// <summary>Проверка правильности имени файла</summary>
        Ссылка: 0
        public bool IsValidLogFileName(string fileName)
        {
            //читать конфигурационный файл
            //вернуть true
            //если конфигурация поддерживается
            return true;
        }
    }
}
```

3. Выполнить тесты.

Зафиксируйте исходное состояние тестов и тестируемого класса в отчет: делайте снимки тестируемого кода, тестов, окна «Результаты тестов» и сохраните в документе MS Word.

##### 2. Рефакторинг проекта для повышения тестопригодности: выделение в отдельный класс операций с файловой системой

Поскольку в тестируемом методе присутствует код, обращающийся к внешним ресурсам, то создание автономных тестов для них невозможно. Для автономных тестов необходимо провести рефакторинг кода. Первым шагом является определение кода, зависящего от внешних ресурсов и выделение его в отдельный класс или метод.

Выделите псевдокод в отдельный класс `FileExtensionManager` и метод `IsValid`. Для этого:

1. Наберите в методе `LogAnalyzer.IsValidLogFileName` после псевдокода создание объекта код создания `FileExtensionManager` и вызов метода `IsValid`. Поскольку данного класса и метода еще не существует при компиляции будет ошибка.

```
//читать конфигурационный файл
//вернуть true
//если конфигурация поддерживается
FileExtensionManager mgr = new FileExtensionManager();
return mgr.IsValid(fileName);
```

Зафиксируйте скриншот в отчете.

2. Установите курсор на `FileExtensionManager` и выберите: Быстрые действия и рефакторинг -> Создать class в новом файле.
3. Установите курсор на `mgr.IsValid` и выберите: Быстрые действия и рефакторинг -> Создать метод.
4. Перенести в созданный метод псевдокод. Вызов исключения `NotImplementedException` оставьте, чтобы проект компилировался.
5. Добавьте summary для класса и метода

На каждом шаге делайте снимки исходного кода, тестов, окна «Результаты тестов» и сохраните в документе MS Word.

Ожидаемый результат:

```

namespace KTR04310.Ivanov.Lib.src.LogAn
{
    /// <summary>Анализатор лог. файлов</summary>
    Ссылка: 0
    public class LogAnalyzer
    {
        /// <summary>Проверка правильности имени файла</summary>
        Ссылка: 0
        public bool IsValidLogFileName(string fileName)
        {
            FileExtensionManager mgr = new FileExtensionManager();
            return mgr.IsValid(fileName);
        }
    }
}

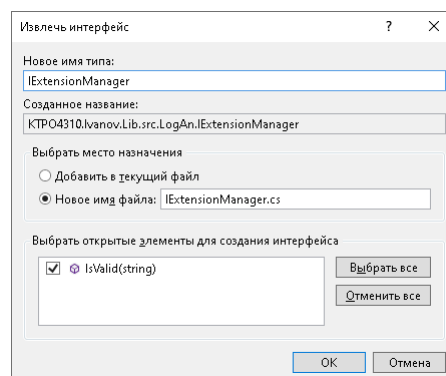
namespace KTR04310.Ivanov.Lib.src.LogAn
{
    /// <summary>Менеджер расширений файлов</summary>
    Ссылка: 2
    internal class FileExtensionManager
    {
        /// <summary>Проверка правильности расширения</summary>
        Ссылка: 1
        internal bool IsValid(string fileName)
        {
            //читать конфигурационный файл
            //вернуть true
            //если конфигурация поддерживается
            throw new NotImplementedException();
        }
    }
}

```

### 3. Рефакторинг проекта для повышения тестопригодности: выделение интерфейса

После первого шага рефакторинга тестируемый код по-прежнему зависит от внешнего ресурса, но опосредованно, через конкретного класс. На втором шаге сделаем так, чтобы тестируемый код манипулировал абстракцией. И создадим класс подделку. Для этого:

1. Замените у класса `FileExtensionManager` и его метода `IsValid` модификатор доступа `internal` на `public`.
2. Выделите интерфейс `FileExtensionManager`, создать его в отдельном файле: Быстрые действия и рефакторинг->Извлечение интерфейса



Имя интерфейса: `IExtensionManager`

3. Объявите переменную диспетчера расширений в тестируемом методе `IsValidLogFileName` как интерфейс.

```
namespace KTO4310.Ivanov.Lib.src.LogAn
{
    /// <summary>Анализатор лог. файлов</summary>
    Ссылка: 0
    public class LogAnalyzer
    {
        /// <summary>Проверка правильности имени файла</summary>
        Ссылка: 0
        public bool IsValidLogFileName(string fileName)
        {
            IExtensionManager mgr = new FileExtensionManager();
            return mgr.IsValid(fileName);
        }
    }
}
```

Зафиксируйте снимки исходного код в документе MS Word.

#### 4. Внедрение зависимости. Внедрение подделки через конструктор.

1. Объявите в классе `LogAnalyzer` поле типа `IExtensionManager`, конструктор с параметром тоже типа, инициализирующий это поле. В методе `IsValidLogFileName` замените создание нового класса на обращение к полю.
2. Создайте заглушку `FakeExtensionManager`, реализующую интерфейс `IExtensionManager`. Разместите исходный код поддельного класса в том же файле, что и тестовый класс.

```
using KTO4310.Ivanov.Lib.src.LogAn;
using NUnit.Framework;
using System;

namespace KTO4310.Ivanov.UnitTest.src.LogAn
{
    [TestFixture]
    Ссылка: 0
    public class LogAnalyzerTests
    {
    }

    /// <summary>Поддельный менеджер расширений</summary>
    Ссылка: 0
    internal class FakeExtensionManager : IExtensionManager
    {
        /// <summary>Это поле позволяет настроить
        /// поддельный результат для метода IsValid</summary>
        public bool WillBeValid = false;

        Ссылка: 2
        public bool IsValid(string fileName)
        {
            return WillBeValid;
        }
    }
}
```

3. Создайте тестовый метод `IsValidFileName_NameSupportedExtension_ReturnsTrue` для тестового сценария «Для поддерживаемых расширение метод возвращает true». Создайте и настройте заглушку для этого сценария. Используйте ее при создании экземпляра `LogAnalyzer`. Добавьте вызов метода `IsValidLogFileName` и проверку результата.

```
[Test]
Ссылка: 0
public void IsValidFileName_NameSupportedExtension_ReturnsTrue()
{
    //Подготовка теста
    FakeExtensionManager fakeManager = new FakeExtensionManager();
    fakeManager.WillBeValid = true;

    LogAnalyzer log = new LogAnalyzer(fakeManager);

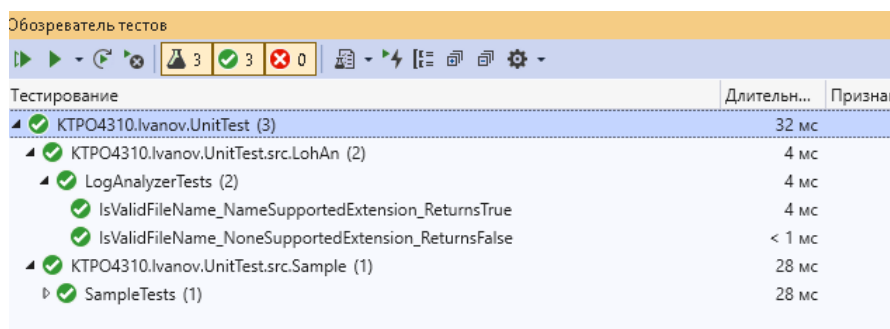
    //Воздействие на тестируемый объект
    bool result = log.IsValidLogFileName("short.ext");

    //Проверка ожидаемого результата
    Assert.True(result);
}
```

4. Добавьте самостоятельно тестовый метод для сценария «Для не поддерживаемых расширение метод возвращает false»
5. Прогоните тесты.

На каждом шаге делайте снимки исходного кода, тестов, окна «Результаты тестов» и сохраните в документе MS Word.

Ожидаемый результат:



Тестирование	Длительн...
KTPO4310.Ivanov.UnitTest (3)	32 мс
KTPO4310.Ivanov.UnitTest.src.LohAn (2)	4 мс
LogAnalyzerTests (2)	4 мс
IsValidFileName_NameSupportedExtension_ReturnsTrue	4 мс
IsValidFileName_NoneSupportedExtension_ReturnsFalse	< 1 мс
KTPO4310.Ivanov.UnitTest.src.Sample (1)	28 мс
SampleTests (1)	28 мс

## 5. Имитация возникновения исключения во внешнем классе.

Пусть необходимо реализовать требование: если в диспетчере расширений файлов возникло исключение, то метод `IsValidFileName` должен вернуть true. Для того чтобы протестировать этот сценарий необходимо, чтобы заглушку можно было настроить для имитации возникновения исключений.

1. Добавьте в класс `FakeExtensionManager` свойство `Exception WillThrow`

2. Доработайте метод IsValid заглушки таким образом, что если поле WillThrow инициализировано, то вызывается данное исключение.

```

/// <summary>Поддельный менеджер расширений</summary>
Ссылка: 6
internal class FakeExtensionManager : IExtensionManager
{
    /// <summary>Это поле позволяет настроить
    /// поддельный результат для метода IsValid</summary>
    public bool WillBeValid = false;

    /// <summary>Это поле позволяет настроить
    /// поддельное исключение вызываемое в методе IsValid</summary>
    public Exception WillThrow = null;

    Ссылка: 2
    public bool IsValid(string fileName)
    {
        if (WillThrow != null)
        {
            throw WillThrow;
        }

        return WillBeValid;
    }
}

```

3. Создайте тестовый метод для сценария «Если диспетчер расширений вызвал исключение, вернуть false»:

IsValidFileName\_ExtManagerThrowsException\_ReturnsFalse

4. Выполните тесты. Каков результат? Ожидаемый результат:

Обозреватель тестов

Тестирование

Тестирование	Длительн...	Признак
KTPO4310.Ivanov.UnitTest (4)	36 мс	❌
KTPO4310.Ivanov.UnitTest.src.LohAn (3)	22 мс	❌
LogAnalyzerTests (3)	22 мс	❌
IsValidFileName_ExtManagerThrowsException_ReturnsFalse	21 мс	❌
IsValidFileName_NameSupportedExtension_ReturnsTrue	1 мс	✅
IsValidFileName_NoneSupportedExtension_ReturnsFalse	< 1 мс	✅
KTPO4310.Ivanov.UnitTest.src.Sample (1)	14 мс	✅

5. Доработайте тестируемый метод, так , чтобы новый тест стал проходить.

6. Выполните тесты.

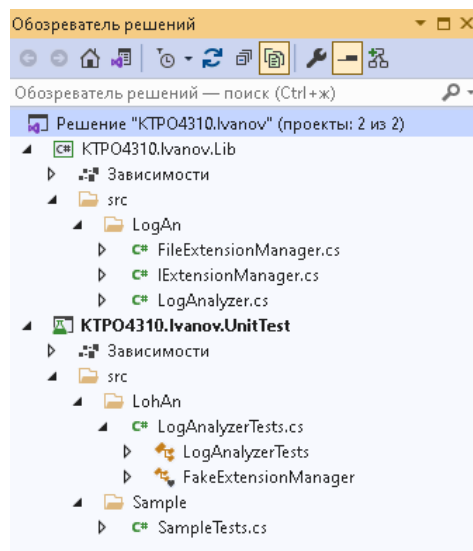
Обозреватель тестов

Тестирование

Тестирование	Длительн...	Признаки
KTPO4310.Ivanov.UnitTest (4)	32 мс	✅
KTPO4310.Ivanov.UnitTest.src.LohAn (3)	4 мс	✅
LogAnalyzerTests (3)	4 мс	✅
IsValidFileName_ExtManagerThrowsException_ReturnsFalse	4 мс	✅
IsValidFileName_NameSupportedExtension_ReturnsTrue	< 1 мс	✅
IsValidFileName_NoneSupportedExtension_ReturnsFalse	< 1 мс	✅
KTPO4310.Ivanov.UnitTest.src.Sample (1)	28 мс	✅

На каждом шаге делайте снимки исходного кода, тестов, окна «Результаты тестов» и сохраните в документе MS Word.

Ожидаемое состояние решение:



## 6. Внедрение зависимости. Внедрение подделки через фабрику.

В предыдущих упражнениях поддельный объект создавался до создания тестируемого класса, и передавался в него уже готовый. Рассмотрим подход, когда тестируемый класс сам запрашивает создание необходимого объекта непосредственно перед обращением к нему. Воспользуемся паттерном проектирования «Фабрика» (Factory), который описывает ситуацию, когда за создание объектов отвечает отдельный класс.

1. Сделайте резервную копию проекта.
2. Создайте в проекте Lib в папке LogAn фабричный класс ExtensionManagerFactory. Для упрощения воспользуемся статическим классом.

Объявите в нем метод Create отвечающий за создание объекта, и метод SetManager, позволяющий внедрить в фабрику поддельный объект.

```

namespace KTO4310.Ivanov.Lib.src.LogAn
{
    /// <summary>Фабрика диспетчеров расширений файлов</summary>
    Ссылка: 0
    public static class ExtensionManagerFactory
    {
        private static IExtensionManager customManager = null;

        /// <summary>Создание объектов</summary>
        Ссылка: 0
        public static IExtensionManager Create()
        {
            if (customManager != null)
            {
                return customManager;
            }

            return new FileExtensionManager();
        }

        /// <summary>Метод позволит тестам контролировать,
        /// что возвращает фабрика</summary>
        /// <param name="mgr"></param>
        Ссылка: 0
        public static void SetManager(IExtensionManager mgr)
        {
            customManager = mgr;
        }
    }
}

```

3. В классе LogAnalyzer удалите параметры конструктора и поле для диспетчера расширений. Диспетчер расширений получите из фабрики. Тесты перестанут компилироваться.

Зафиксируйте это в отчете в виде кода тестов и списка ошибок.

4. Исправьте тесты:

добавьте конфигурирование фабричного класса поддельным диспетчером расширений с помощью метода SetManager.

```

[Test]
Ссылка: 0
public void IsValidFileName_NameSupportedExtension_ReturnsTrue()
{
    //Подготовка теста
    FakeExtensionManager fakeManager = new FakeExtensionManager();
    fakeManager.WillBeValid = true;
    //...конфигурируем фабрику для создания поддельных объектов
    ExtensionManagerFactory.SetManager(fakeManager);

    LogAnalyzer log = new LogAnalyzer();

    //Воздействие на тестируемый объект
    bool result = log.IsValidLogFileName("short.ext");

    //Проверка ожидаемого результата
    Assert.True(result);
}

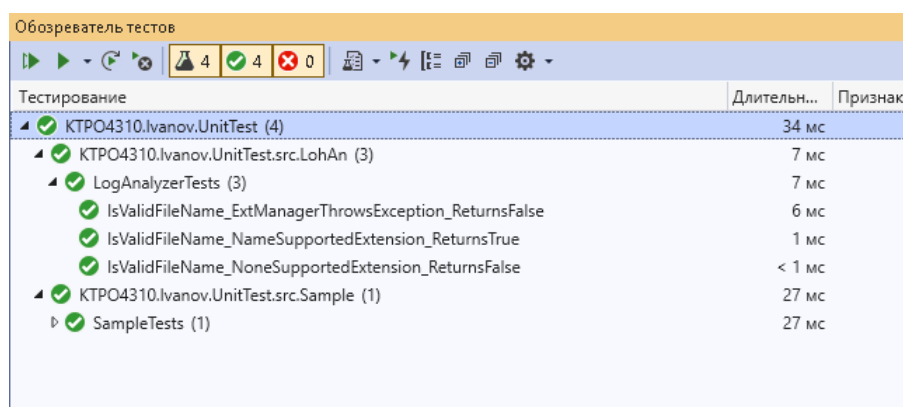
```

5. Поскольку в тесте изменяли глобальное состояние системы, необходимо выполнять возврат в исходное состояние после каждого теста. Для этого добавьте в тестовый класс метод AfterEachTest отмеченный атрибутом [TearDown]. В данном методе вызовите метод SetManager фабричного класса со значением null.

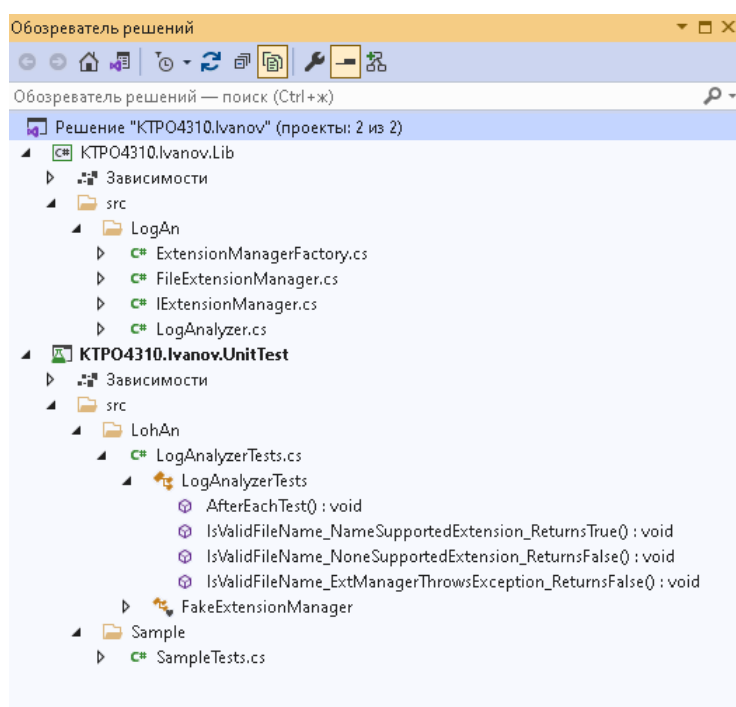


## 6. Выполните тесты.

На каждом шаге делайте снимки исходного кода, тестов, окна «Результаты тестов» и сохраните в документе MS Word.



Ожидаемое состояние решения:



## 7. Реализовать диспетчер расширений и клиентское приложение.

1. Добавьте в решение новый проект консольного приложения.

Имя проекта укажите КТРО4310.Ivanov.Service

2. Организуйте проект согласно принятым ранее соглашениям.

3. Добавьте в проект файл настроек. Добавьте в него информацию о правильно расширении.

4. Реализуйте логику класса FileExtensionManager, так, чтобы допустимое расширение читалось из файла настроек.

5. Добавьте в КТРО4310.Ivanov.Service вызов класса LogAnalyzer для разных имен файлов, с правильными и не правильными расширениями. Результат вызова выведете на консоль.

6. Протестируйте и отладьте приложение.

7. Запустите тесты.

Делайте снимки исходного кода, тестов, окна «Результаты тестов», результат запуска консольного приложения и сохраните в документе MS Word.

### **Содержание отчета**

1. Постановка задачи.
2. Экранные формы с результатами выполнения заданий.
3. Выводы.