

Лабораторная работа №3.

Тестирование взаимодействия и подставные объекты

Цель работы

Приобретение практических навыков использования подставных объектов, для тестирования взаимодействия при автономном тестировании модулей, практика использования тестового каркаса NUnit, практика внедрения зависимости.

Порядок выполнения работы

1. Подготовка проекта

Для выполнения данной лабораторной работы возьмите решение, полученной в результате выполнения лабораторной работы №2.

Выполните тесты.

Зафиксируйте исходное состояние окна «Обозреватель решения», тестируемого класса и тестов окна «Результаты тестов»: и сохраните в документе MS Word.

2. Использование подставного объекта

Пусть в классе LogAnalyzer определен метод Analyze, осуществляющий анализ файла. И это метод должен передавать внешней службе сообщение, когда имя файла слишком короткое. Но сама служба, взаимодействие с которой нужно протестировать еще не готова, обращение к ней представлено псевдокодом. Поэтому необходимо подвергнуть проект рефакторингу и создать интерфейс, для которого можно будет создать поддельный объект.

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

1. Добавьте в класс LogAnalyzer новый метод Analyze, код для проверки длины файла и взаимодействия с веб-сервисов, как показано на рисунке.

```

/// <summary>Анализатор лог. файлов</summary>
Ссылка: 10
public class LogAnalyzer
{
    /// <summary>Проверка правильности имени файла</summary>
    Ссылка: 5
    public bool IsValidLogFileName(string fileName) {...}

    /// <summary>Анализировать лог. файл </summary>
    /// <param name="fileName"></param>
    Ссылка: 1
    public void Analyze(string fileName)
    {
        if (fileName.Length < 8)
        {
            //Передать внешней службе сообщение об ошибке
            //""Слишком короткое имя файла: " + fileName
        }
    }
}

```

2. Выделите интерфейс IWebService с одним методом

void LogError(string message);

Это интерфейс можно будет использовать как для создания заглушек, так и подставок.

3. Замените псевдокод на обращение к веб-службе через интерфейс IWebService. Внедрите зависимость используя рассмотренный в лабораторной работе №2 метод внедрения зависимости через фабрику. Воспользуйтесь методом [TearDown] для восстановления исходного состояния фабрики.

4. Создайте поддельный объект FakeWebService: IWebService.

Этот объект выглядит как заглушка, но содержит дополнительный код, который позволяет сохранить и затем проверить состояние объекта после вызова метода, и сделать утверждение о том, что объект вызван верно. Таким образом объект можно будет использовать в качестве подставки.

```

/// <summary>Поддельная веб-служба </summary>
Ссылка: 0
internal class FakeWebService : IWebService
{
    /// <summary>Это поле запоминает состояние после вызова метода LogError
    /// при тестировании взаимодействия утверждения высказываются относительно</summary>
    public string LastError;

    Ссылка: 2
    public void LogError(string message)
    {
        LastError = message;
    }
}

```

5. Создайте тестовый метод для сценария «Если имя слишком короткое, вызываем веб службу».

Analyze_TooShortFileName_CallsWebService()

Обратите внимание, что утверждение высказывается относительно подставного объекта, а не тестируемого метода. Это происходит потому, что тестируется взаимодействие LogAnalyzer с веб службой. Префикс mock в имени переменной для поддельного объекта указывает на то, что поддельный объект используется в качестве подставки.


```
[Test]
public void Analyze_TooShortFileName_CallsWebService()
{
    //Подготовка теста
    FakeWebService mockWebService = new FakeWebService();
    WebServiceFactory.SetWebService(mockWebService);
    LogAnalyzer log = new LogAnalyzer();
    string tooShortFileName = "abc.ext";

    //Воздействие на тестируемый объект
    log.Analyze(tooShortFileName);

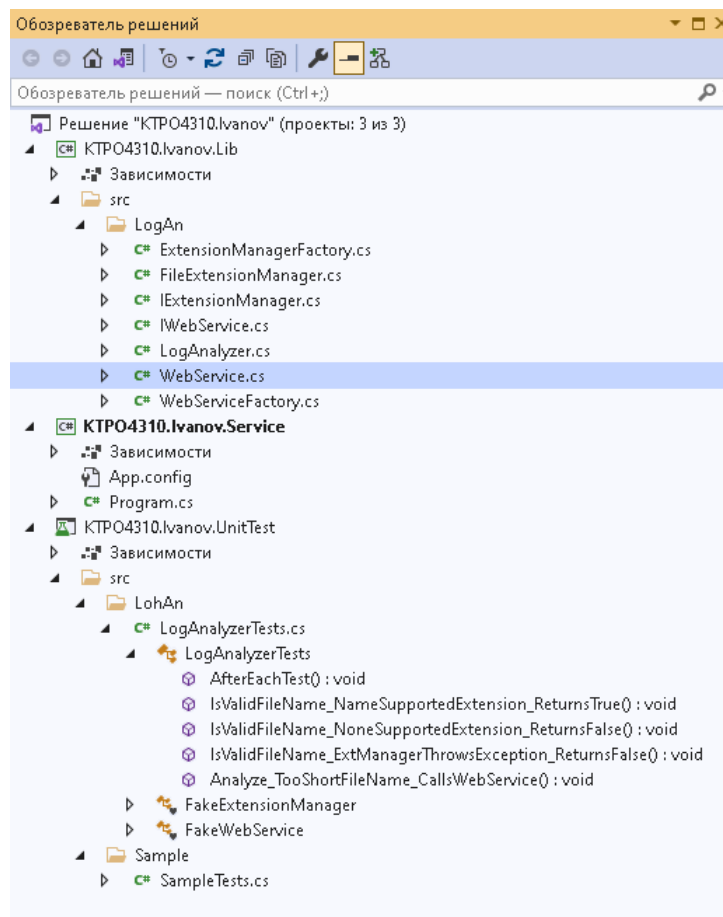
    //Проверка ожидаемого результата
    StringAssert.Contains("Слишком короткое имя файла: abc.ext",
        mockWebService.LastError);
}
```

6. Выполните тесты.

Ожидаемый результат:

Обозреватель тестов		
		
Тестирование	Длительн...	Пг
KTPO4310.Ivanov.UnitTest (5)	38 мс	
KTPO4310.Ivanov.UnitTest.src.LohAn (4)	10 мс	
LogAnalyzerTests (4)	10 мс	
Analyze_TooShortFileName_CallsWebService	9 мс	
IsValidFileName_ExtManagerThrowsException_ReturnsFalse	1 мс	
IsValidFileName_NameSupportedExtension_ReturnsTrue	< 1 мс	
IsValidFileName_NoneSupportedExtension_ReturnsFalse	< 1 мс	
KTPO4310.Ivanov.UnitTest.src.Sample (1)	28 мс	
SampleTests (1)	28 мс	

Ожидаемая структура решения:



3. Несколько поддельных объектов. Совместное использование поставки и заглушки

Пусть теперь класс LogAnalyzer должен обратиться к веб-службе, и если служба вернет ошибку, то записать эту ошибку в другую зависимость, отправив сообщение по электронной почте. Логика выглядит следующим образом:

```
//Если имя слишком короткое
if (fileName.Length < 8)
{
    try
    {
        //Передать внешней службе сообщение об ошибке
        IWebService srv = WebServiceFactory.Create();
        srv.LogError("Слишком короткое имя файла: " + fileName);
    }
    catch (Exception e)
    {
        //Отправить сообщение по электронной почте
        //Email.SendEmail("someone@somewhere.com", "Невозможно вызвать веб-сервис", e.Message);
    }
}
```

Для того чтобы протестировать это поведение, нам понадобится имитировать исключение при вызове веб службы, т.е. понадобится заглушка. Для проверки взаимодействия с почтовой службы понадобится подставка.

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

1. Для заглушки веб-службы воспользуемся интерфейсом и поддельным объектом, созданным в предыдущем упражнении. Доработайте класс FakeWebService, так, чтобы можно было управлять результатом вызова, а и именно имитировать вызов исключения, так как это было сделано в лабораторной работе №2.
2. Выделите интерфейс IEmailService с одним методом
void SendEmail(string to, string subject, string body),
где to – адрес, subject – тема, body – сообщение.
Это интерфейс будем использовать как для создания подставки.
3. Замените псевдокод на обращение к веб-службе через интерфейс IEmailService. Внедрите зависимость используя рассмотренный в лабораторной работе №2 метод внедрения зависимости через фабрику. Поскольку настоящего класса службы еще нет, в метод фабрики вместо его создания можем разместить код:

```
//Настоящая почтовая служба еще не реализована  
throw new NotImplementedException();
```

Воспользуйтесь методом [TearDown] для восстановления исходного состояния фабрики.

4. Создайте поддельный объект FakeEmailService: IEmailService.
Добавьте код, который позволить делать утверждения относительно этого объекта, после вызова метода SendMail. Необходимо сохранить состояние всех параметров метода, так как утверждения тоже должно высказываться относительно правильности всех трех значений.
5. Создайте тестовый метод для сценария «Если веб-служба вызывает исключение, отправляем почтовое сообщение».

Analyze_WebServiceThrows_SendsEmail()

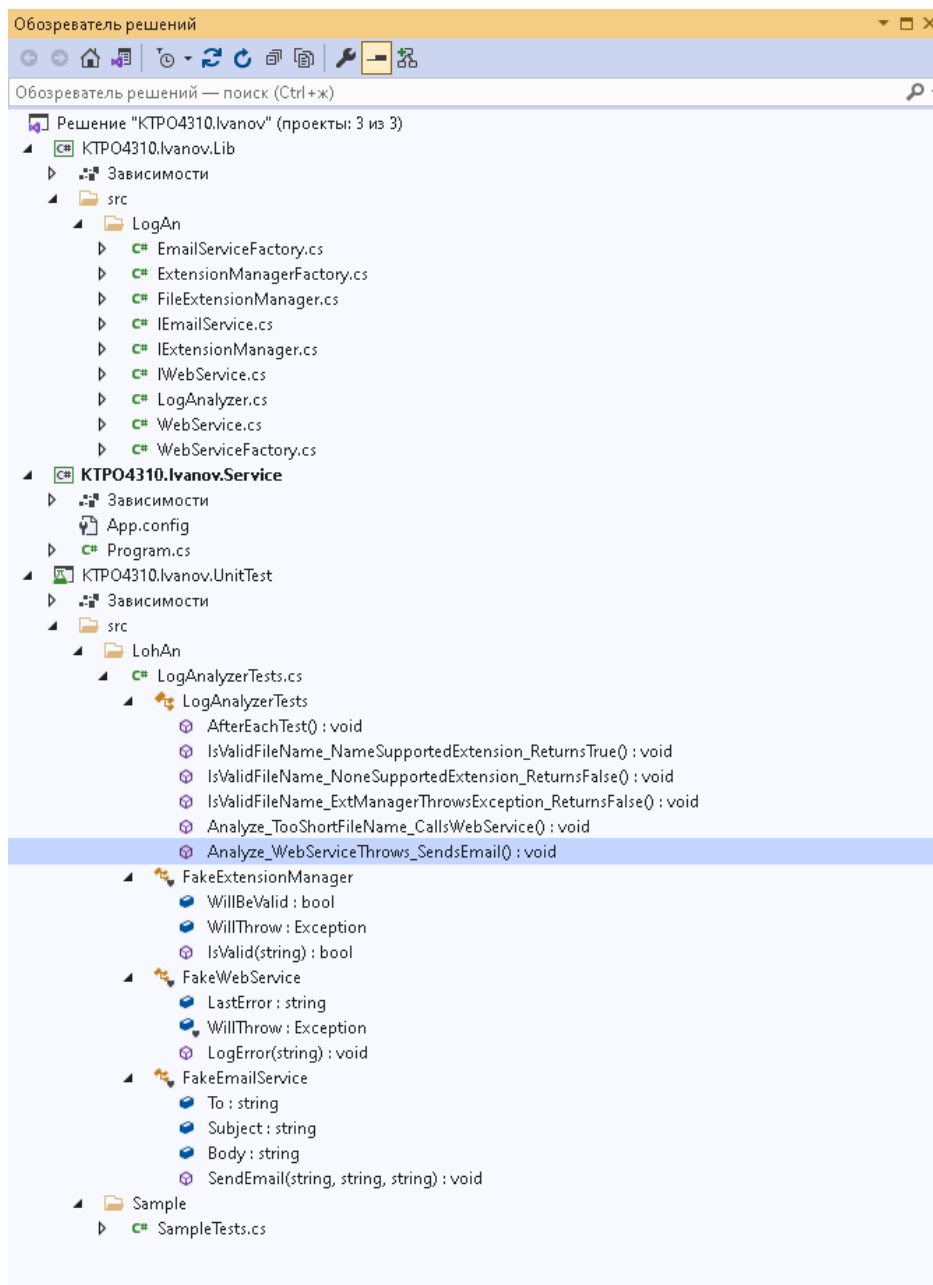
Обратите внимание на именование переменных для поддельного объекта. Префикс stub указывает, что объект является заглушкой.

1

- Восстановите правильный код.

Ожидаемый результат.

Ожидаемая структура решения:



Содержание отчета

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