Лабораторная работа №5.
Тестирование событий

Цель работы

Приобретение практических навыков использования делегатов (delegate) и событий (event); тестирования операций, связанных с событием; практика использования тестового каркаса NUnit, практика использования изолирующего каркаса NSubstitute.

Задание на лабораторную работу

1. Подготовить учебный проект
2. Реализовать генерацию события
3. Реализовать подписку на событие
4. Реализовать тесты для проверки прослушивателя события, с имитацией события в ручной заглушке, созданный с помощью наследования.
5. Реализовать тесты для проверки прослушивателя события с генерацией имитацией события в с помощью NSubstitute
6. Тестирования факта генерации события с использованием анонимного делегата
7. На каждом шаге делайте снимки исходного кода создаваемых или изменяемых классов и тестов, окна «Результаты тестов» и «Обозреватель решения» и сохраните в документе MS Word.
8. Оформить отчет

Порядок выполнения работы

1. Подготовка проекта

Для выполнения данной лабораторной работы возьмите решение, полученное в результате выполнения лабораторной работы №4.

Выполните тесты.

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

2. Генерация события

Пусть, согласно требованиям, необходимо обеспечить возможность, чтобы класс LogAnalyzer мог уведомлять вызывающий код о ходе выполнения анализа файла. Например, для отображения прогресса выполнения обработки. Реализуем это требования с помощью событий.

1. Объявите в проекте «.Lib» делегат LogAnalyzerAction. Разметите его в отдельном файле «.cs» файл в папке LogAn.

```
/// <summary>Обьявление делегата</summary>
public delegate void LogAnalyzerAction();
```

2. Объявите событие с помощью ключевого слова event типа LogAnalyzerAction.

```
/// <summary>Анализато лог. файлов</summary>
Ссылок: 22
public class LogAnalyzer
{
    /// <summary>Обьявление события </summary>
    public event LogAnalyzerAction Analyzed = null;
```

3. Добавьте вызов события в метод LogAnalyzer. Analyze. Обратите внимание, вызов события необходимо выполнять только, если для него есть обработчик.

```
/// <summary>Анализировать лог. файл </summary>
/// <param name="fileName"></param>
Ссылок: 4 | 0/4 пройдены
public void Analyze(string fileName)
{
    //Если имя слишко короткое
    if (fileName.Length < 8)...

    //Обработка лога
    //..

    //Вызов события
    if (Analyzed != null)
    {
        Analyzed();
    }

}
```

4. Постройте приложение и выполните тесты.

3. Подписка объекта на событие

Пусть за подготовку данных для отображения отвечает объект Презентатор (Presenter), а за отображение класс Представление (View). Презентатор будет получать событие, формировать вывод, и передавать его Представлению для отображения.

1. Добавьте в проекте «.Lib» в папке LogAn интерфейс IView для Представления . Включите в него один метод Render:

```
Ссылок: 0
void Render(string text);
```

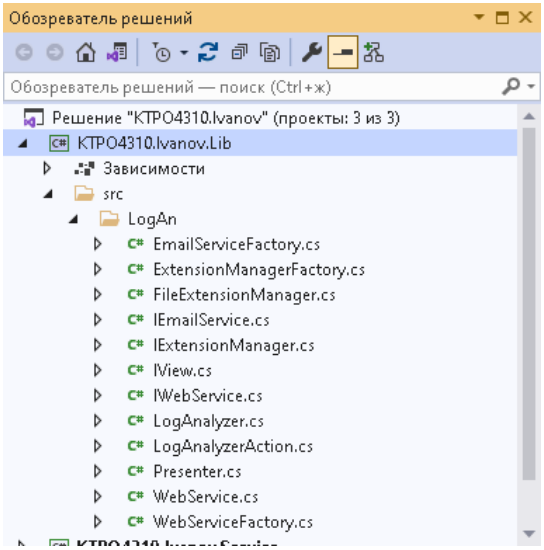
2. Добавьте в проекте «.Lib» в папке LogAn класс Presenter – Презентатор
3. В классе Presenter объявите конструктор с двумя параметрами:
   (LogAnalyzer logAnalyzer, IView view)
4. Добавьте два поля для объектов LogAnalyzer и IView
   и инициализируйте их в конструкторе значениями из параметров
5. Объявите в классе приватный метод OnLogAnalyzed()
   Это обработчик события. По сигнатуре он должен совпадать с сигнатурой делегата LogAnalyzerAction. Обратите внимание на соглашение о наименовании метода. Добавьте в него вызов обращение к Представлению.

```
Ссылок: 0
private void OnLogAnalyzed()
{
    view.Render("Обработка завершена");
}
```

6. Добавьте в конструктор подписку на событие:

```
logAnalyzer.Analyzed += OnLogAnalyzed;
```

7. Постройте приложение и выполните тесты.
8. Ожидаемый результат:



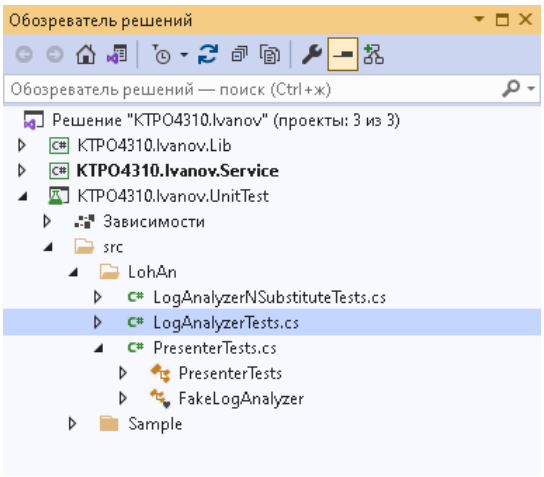
4. Тестирование прослушивания события. Введение зазора с помощью наследования.

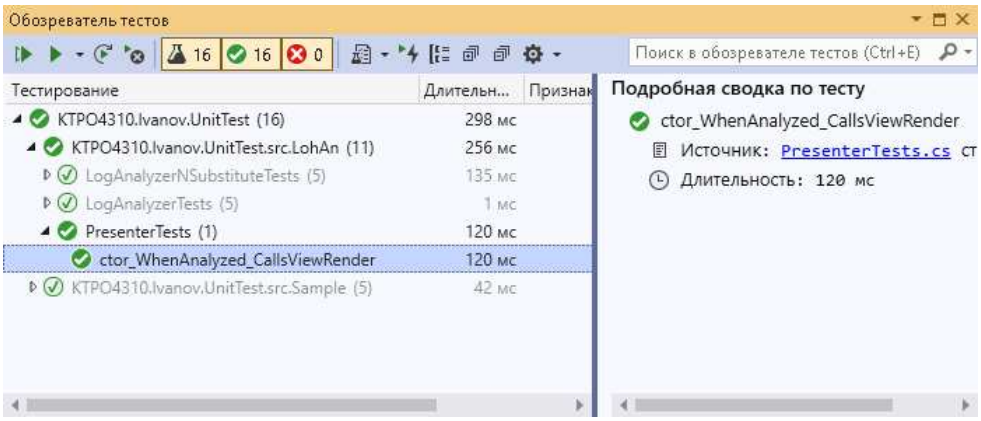
В данном сценарии необходимо проверить, что один объект зарегистрирован в качестве получателя события, генерируемого другим объектом. Для этого нужно убедиться, что объект-прослушиватель что-то делает при возникновении события. Для тестирования необходима возможность вызывать событие, не вызывая метод LogAnalyze.Analyze(), в котором оно генерируется согласно логике программы. Так как вызвать событие можно только из того класса, в котором оно объявлено, нам необходим объект заглушка для LogAnalyze, умеющий генерировать событие по требованию теста. А также ввести зазор, чтобы можно было подменить настоящий объект на поддельный. В данном упражнение воспользуемся наследованием.

1. Добавьте в проет «.UnitTest» в папку LogAn тестовый класс PresenterTests.
2. Выделите код вызова события в метод RaiseAnalyzedEvent(). Сделайте его защищенным.
3. Добавьте в файл PresenterTests.cs поддельный объект, унаследовав его от LogAnalyzer. Добавьте в него открытый метод, в котором вызовите метод RaiseAnalyzedEvent() базового класса.

```
/// <summary>Заглушка для имитации вызова события </summary>
Ссылок: 0
class FakeLogAnalyzer : LogAnalyzer
{
    Ссылок: 0
    public void CallRaiseAnalyzedEvent()
    {
        base.RaiseAnalyzedEvent();
    }

}
```

4. Добавьте тестовый метод для сценария «Если вызвано событие, то класс вызывает отображение»:
   ctor_WhenAnalyzed_CallsViewRender
5. Реализуй тест по трем шагам:
   1) Подготовка теста, включающий:
      создание заглушки для LogAnalyze;
      создание подставки для IView с использованием NSubstitute,
      создание тестируемого объекта Presenter
   2) Воздействие на тестируемый объект.
      Заключается в вызове метода заглушки генерации события
   3) Проверка ожидаемого результата
      Проверка, что для подставки IView вызван метод Render и его параметр содержит значение «Обработка завершена»
6. Выполните тесты:
7. Ожидаемый результат:





5. Имитация события с помощью NSubstitute

В этом упражнении реализуем тест для прослушивателя события, аналогичный упражнению 4. Но вместо заглушки для LogAnalyzer созданной вручную воспользуемся каркасом NSubstitute

1. Выделите для класса LogAnalyzer интерфейс ILogAnalyze. Включите в него все методы и событие.
2. В классе Presenter замените использование класса LogAnalyzer на интерфейс.
3. Постройте решение и выполните тесте. Зафиксируйте результат.
4. Создайте в PresenterTests тестовый метод ctor_WhenAnalyzed_CallsViewRender_NSubstitute

   По сравнение с предыдущим тестом замените создание заглушки на использование NSubstitute:

```
ILogAnalyzer stubLogAnalyzer = Substitute.For<ILogAnalyzer>();
```

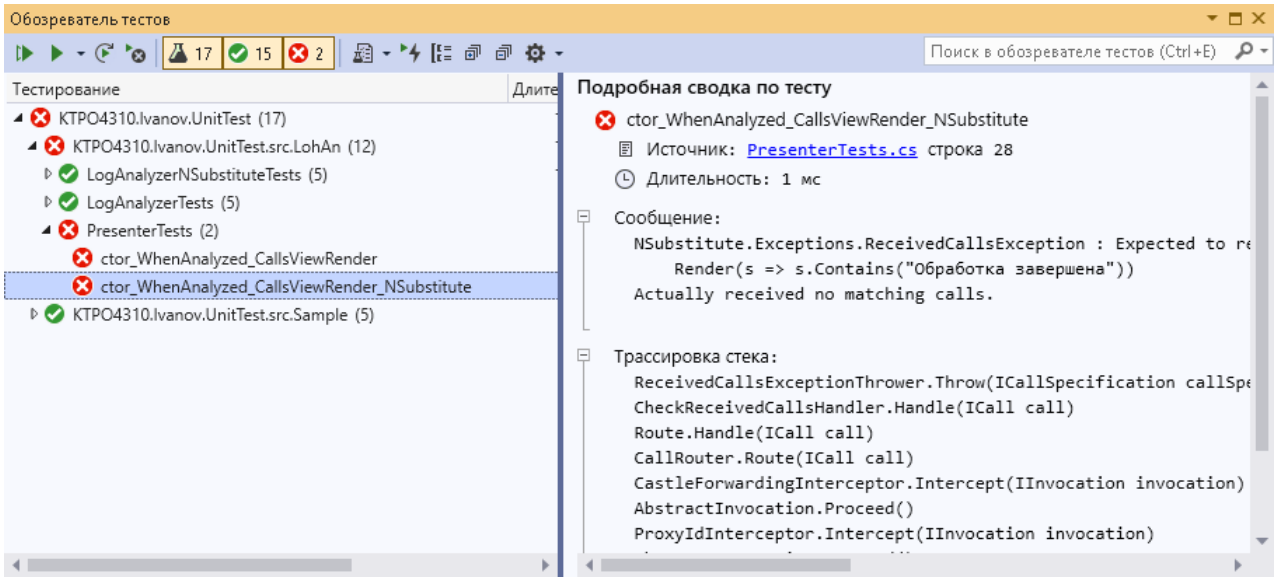
   Вызов исключения замените на конструкцию:

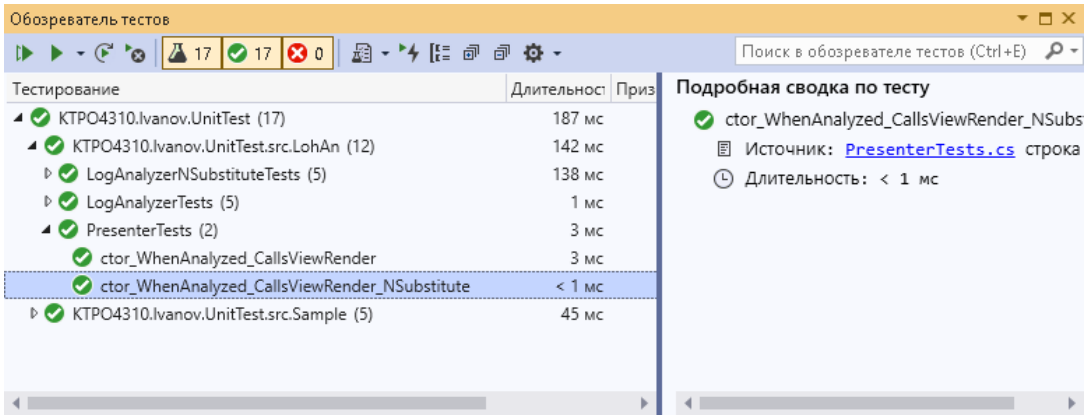
```
//Воздействие на тестируемый объект
mockView.Loaded += Raise.Event<Action>();
```

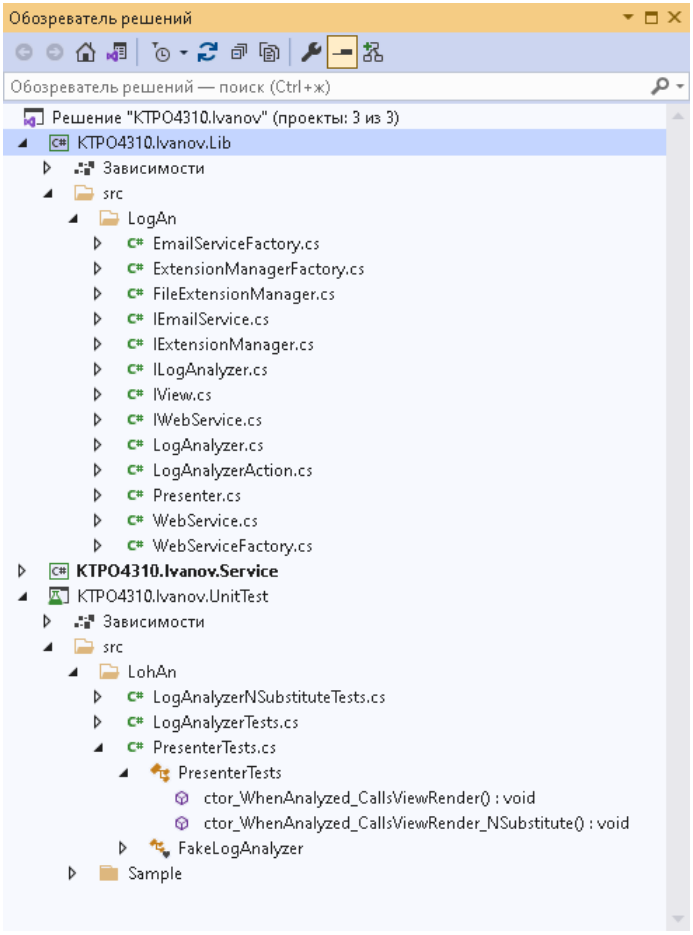
5. Выполните тесты.
6. Проверьте правильность тестовых методов. Для этого внесите в тестируемый метод дефект, ошибку которую должен обнаружить тест. Выполните тест и зафиксируйте результат, в том числе и текст сообщения об ошибке. Убедитесь, что вариант теста и рукописной и динамической заглушкой работают одинаково.

Восстановите правильный код.

7. Ожидаемый результат







6. Тестирования факта генерации события

Для проверки, что тестируемый класс вызывает событие на него нужно подписаться в тестовом классе. Воспользуемся для этого анонимным методом.

1. Создайте тестовый метод для сценария «Если файл проанализирован, то вызывается событие»

   Analyze_WhenAnalyzed_FiredEvent

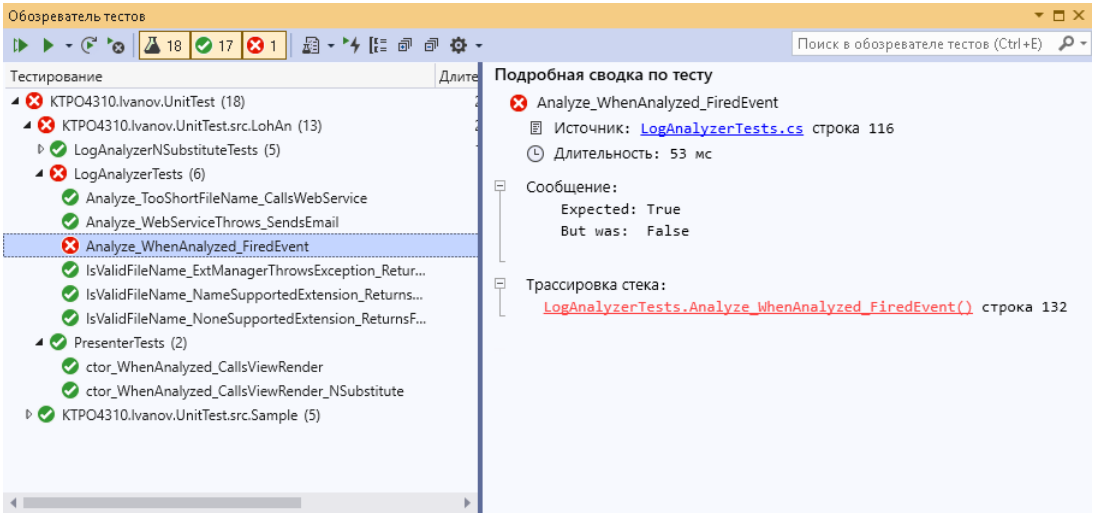
   В каком тестовом классе должен размещаться этот метод?

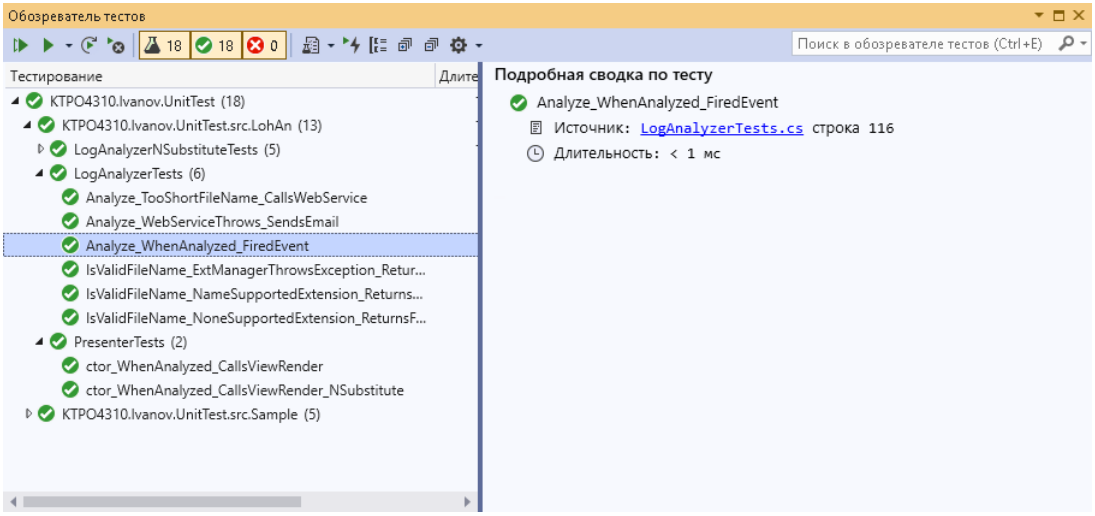
```
[Test]
Ссылок: 0
public void Analyze_WhenAnalyzed_FiredEvent()
{
    //Подготовка теста
    bool analyzedFired = false;

    LogAnalyzer logAnalyzer = new LogAnalyzer();
    //..используем анонимный метод в качестве обрботчика события
    logAnalyzer.Analyzed += delegate ()
    {
        analyzedFired = true;
    };

    //Воздействие на тестируемый обьект
    logAnalyzer.Analyze("validfilename.vld");

    //Проверка ожидаемого результата
    Assert.IsTrue(analyzedFired);
}
```

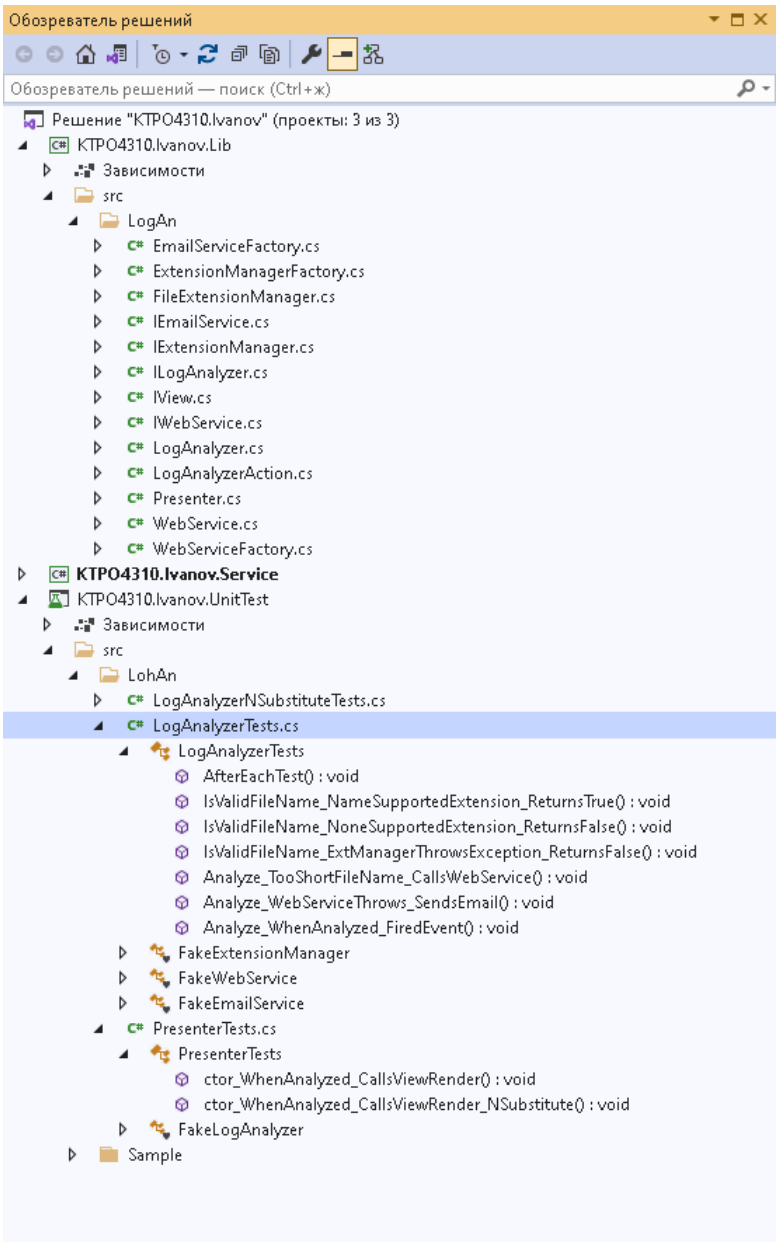
8. Выполните тесты.

9. Проверьте правильность тестового метода. Для этого внесите в тестируемый метод дефект, ошибку которую должен обнаружить тест. Выполните тест и зафиксируйте результат, в том числе и текст сообщения об ошибке. Восстановите правильный код.

10. Ожидаемый результат







Содержание отчета

1. Постановка задачи.
2. Экранные формы с результатами выполнения задания: окна «Обозреватель решения», окна «Обозреватель тестов», исходный код тестов, исходный код тестируемых классов и поддельных объектов.
3. Выводы.