**Лабораторная работа 1**

**Введение в ООП на языке C#**

**TDD – Test Driven Development**

Методология разработки через тестирование. Три закона TDD:

1. Новый рабочий код пишется только после того, как будет написан модульный тест, который не проходит (если код теста не компилируется, считается, что он не проходит).
2. Вы пишете ровно такой объем кода модульного теста, какой необходим для того, чтобы этот тест, проверяющий работу класса, не проходил.
3. Вы пишете ровно такой объем рабочего кода, какой необходим для прохождения модульного теста, который в данный момент не проходит.

Для обеспечения эффективной защиты от ошибок тесты должны выполнять ("покрывать") большую часть кода. Но 100% покрытие тестируемого кода не гарантирует правильность программы.

При разработке программ через TDD примеряют цикл Red–Green–Refactor.

* Red - думайте о тем, что вы хотите разработать (создаются тестовые методы, код не компилируется).
* Green - думайте о том, как сделать так, чтобы программа прошла ваш тест (по вызову тестируемого кода, пишется сигнатура разрабатываемого метода).
* Refactor - думайте о том, как улучшить вашу текущую реализацию (наполняют код разрабатываемого метода таким образом, чтобы тест выполнялся и оптимизируют его).

**Создание UNIT-тестов в Visual Studio**

Модульное тестирование или unit-тестирование – это тестирование отдельных частей программы.

Задача тестирования состоит в том, чтобы написать дополнительный класс с методами, который будет тестировать основные классы программы. Класс для тестирования – это обычный C#-класс, который компилируется в dll-файл, но этот C#-класс должны быть размечен с помощью специальных атрибутов.

Создание теста.

1 способ. В файле класса вызвать контекстное меню и выбрать пункт Создание модульных тестов. Затем нажмите OK. Файл с тестовым классом будет создан.

2 способ. Добавьте новый проект модульного теста с именем MyTest в данное решение. Вместо префикса My запишите имя класса, к которому создаете тесты (рис. 1).

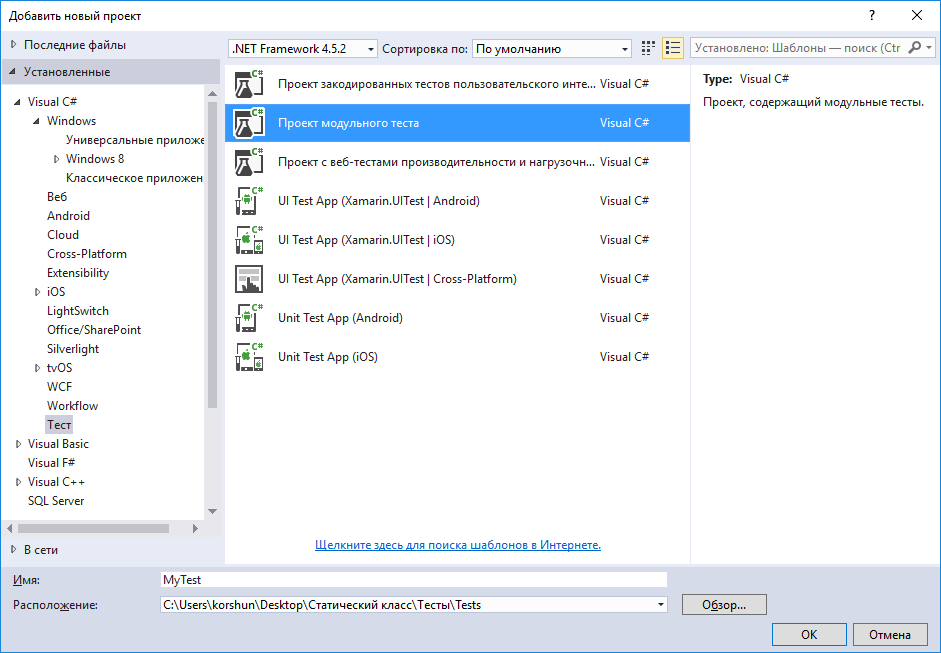


Рисунок 1. Добавление нового проекта

В тестовом проекте необходимо добавить ссылку на основной проект: Свойства проекта -> Добавить -> Ссылка. Проект с тестируемым классом и тест находятся в одном решении, поэтому ссылку нужно искать на вкладке Проекты в Решении (рис. 2).

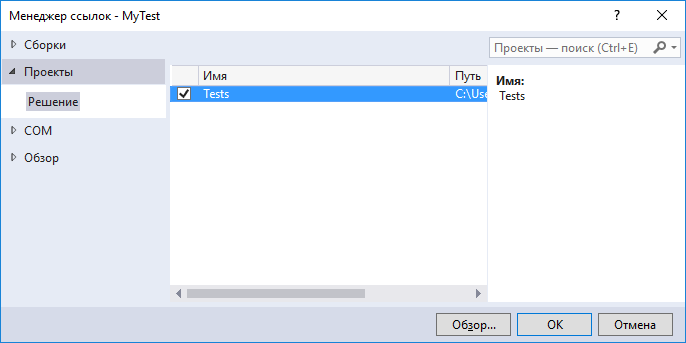


Рисунок 2. Добавление ссылки на тестируемый проект

В тестовом проекте создан тестовый класс

[TestClass]

public class MyTests

{

[TestMethod]

public void TestMethod()

{

//Arrange (Устанавливаем) – производим настройку входных данных для теста.

//код создания объектов, переменных, инициализации значений

...

// Act (Действуем) – выполняем действие, результаты которого тестируем.

//код вызова тестируемого метода

...

//Assert (Проверяем) – проверяем результаты выполнения.

// код вызова Assert

...

}

}

Обязательно объявите тестируемый класс с модификатором public, чтобы он был доступен в других проектах решения.

В разделе ссылок тестового проекта добавьте ссылку на проект с тестируемым классом. В тестовый класс добавьте ссылку на тестируемый проект, используя using.

Для создания тестов используется статический класс Assert, который находится в пространстве имен Microsoft.VisualStudio.TestTools.UnitTesting. В нем определен набор методов, которые можно использовать для проверки работоспособности создаваемых классов.

Если метод разрабатываемого класса выкидывает исключение, то для него должно быть написано несколько тестов: тесты, проверяющие благополучный исход и тесты, обрабатывающие исключительную ситуацию. В случае проверки исключения к тестирующему методу добавляют атрибут:

[ExpectedException(typeof(ТипПроверяемогоИсключения))]

Для проверки выполнения тестов откройте панель Обозреватель тестов (меню Тест –> Окна –> Обозреватель тестов).

**Задачи для самостоятельно решения**

**Задача 1**. **Напишите** класс Indexer, который создается как обертка над массивом double[], и открывает доступ к его подмассиву некоторой длины, начиная с некоторого элемента. **Создайте тестовый проект и добавьте в него данные тесты**:

public class UnitTest1

{

double[] array = new double[] { 1, 2, 3, 4 };

[TestMethod]

public void HaveCorrectLength()

{

var indexer = new Indexer(array, 1, 2);

Assert.AreEqual(2, indexer.Length);

}

[TestMethod]

public void GetCorrectly()

{

var indexer = new Indexer(array, 1, 2);

Assert.AreEqual(2, indexer[0]);

Assert.AreEqual(3, indexer[1]);

}

[TestMethod]

public void SetCorrectly()

{

var indexer = new Indexer(array, 1, 2);

indexer[0] = 10;

Assert.AreEqual(10, array[1]);

}

[TestMethod]

public void IndexerDoesNotCopyArray()

{

var indexer1 = new Indexer(array, 1, 2);

var indexer2 = new Indexer(array, 0, 2);

indexer1[0] = 100500;

Assert.AreEqual(100500, indexer2[1]);

}

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void FailWithWrongArguments1()

{

Assert.Equals(typeof(ArgumentException), new Indexer(array, -1, 3));

}

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void FailWithWrongArguments2()

{

Assert.Equals(typeof(ArgumentException), new Indexer(array, 1, -1));

}

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void FailWithWrongArguments3()

{

Assert.Equals(typeof(ArgumentException), new Indexer(array, 1, 10));

}

[TestMethod]

[ExpectedException(typeof(IndexOutOfRangeException))]

public void FailWithWrongIndexing1()

{

var indexer = new Indexer(array, 1, 2);

Assert.Equals(typeof(IndexOutOfRangeException), indexer[-1]);

}

[TestMethod]

[ExpectedException(typeof(IndexOutOfRangeException))]

public void FailWithWrongIndexing2()

{

var indexer = new Indexer(array, 1, 2);

Assert.AreNotEqual(typeof(IndexOutOfRangeException), indexer[10]);

}

}

Следите за целостностью данных в Indexer.

**Задача 2**.

Используя три закона TDD описать классы для решения СЛУ методом Гаусса. Тесты должны опережать описание класса.

СЛУ состоит из m уравнений от n переменных и имеет вид:



Решить СЛУ означает найти все значения переменных *x*1, *x*2, …, *x*n, при которых каждое уравнение системы превращается в верное равенство.

**Пояснение**. Так как СЛУ состоит из линейных уравнений и для того чтобы решить СЛУ нужно уметь проводить элементарные преобразования, то возникает необходимость описать класс – линейное уравнение (LinearEquation) от n переменных. Коэффициенты линейного уравнения – вещественные числа.

Для класса LinearEquation:

* описать (перегрузить) конструктор с параметром. Предусмотреть создание линейного уравнения из строки с коэффициентами; из массива или списка коэффициентов; линейного уравнения от n переменных с нулевыми коэффициентами.
* описать метод инициализации (задания значений) уравнения случайными числами; одинаковыми значениями.
* перегрузить бинарные операции сложения и вычитания двух линейных уравнений;

**public** **static** **LinearEquation** **operator** **+**(LinearEquation a, LinearEquation b);

**public** **static** **LinearEquation** **operator** **-**(LinearEquation a, LinearEquation b);

* перегрузить операцию умножения линейного уравнения на вещественное число справа и слева;

**public** **static** **LinearEquation** **operator** **\***(LinearEquation a, double r);

**public** **static** **LinearEquation** **operator** **\***(double r, LinearEquation a);

* перегрузить унарную операцию минус (то есть умножение линейного уравнения на -1) для линейного уравнения;

**public** **static** **LinearEquation** **operator** **-**(LinearEquation a);

* перегрузить операторы == и != для сравнения двух линейных уравнений;

**public** **static** **bool** **operator** ==(LinearEquation a, LinearEquation b);

**public** **static** **bool** **operator** !=(LinearEquation a, LinearEquation b);

* переопределить операторы false и true для обозначения противоречивого и разрешимого уравнения. Противоречивое уравнение имеет вид ;

**public** **static** **bool** **operator** **false**(LinearEquation a);

**public** **static** **bool** **operator** **true**(LinearEquation a);

* переопределить метод ToString(); (переопределение данного метода дает возможность выводить LinearEquation в консоль, указывая только имя переменной);
* переопределить неявное преобразование к списку List<double> (или массиву):

**public** **static** **implicit** **operator** **List<double>** (LinearEquation a);

Описать класс СЛУ (SystemOfLinearEquation) от m уравнений с n переменными.

Для класса SystemOfLinearEquation :

* описать конструктор с параметром (параметр – количество переменных);
* перегрузить индексатор [] для обращения к уравнению по его номеру в СЛУ;

**public** **int** **this**[**int** i]

        {

**get** { ; }

**set** { ; }

        };

* реализовать метод приведения СЛУ к ступенчатому виду;
* реализовать решение СЛУ методом Гаусса – получить решение в виде массива или списка значений переменных, в случае неразрешимости СЛУ – создать исключение;
* переопределить метод ToString().