Лабораторная работа №5: Модульное тестирование

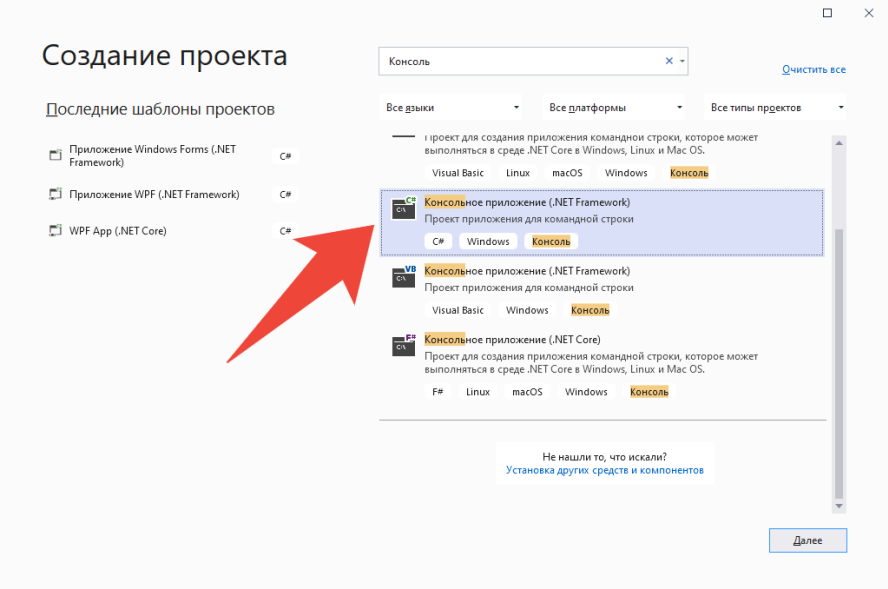
Цель работы:

– получение базовых навыков разработки и реализации модульных тестов в среде NUnit.

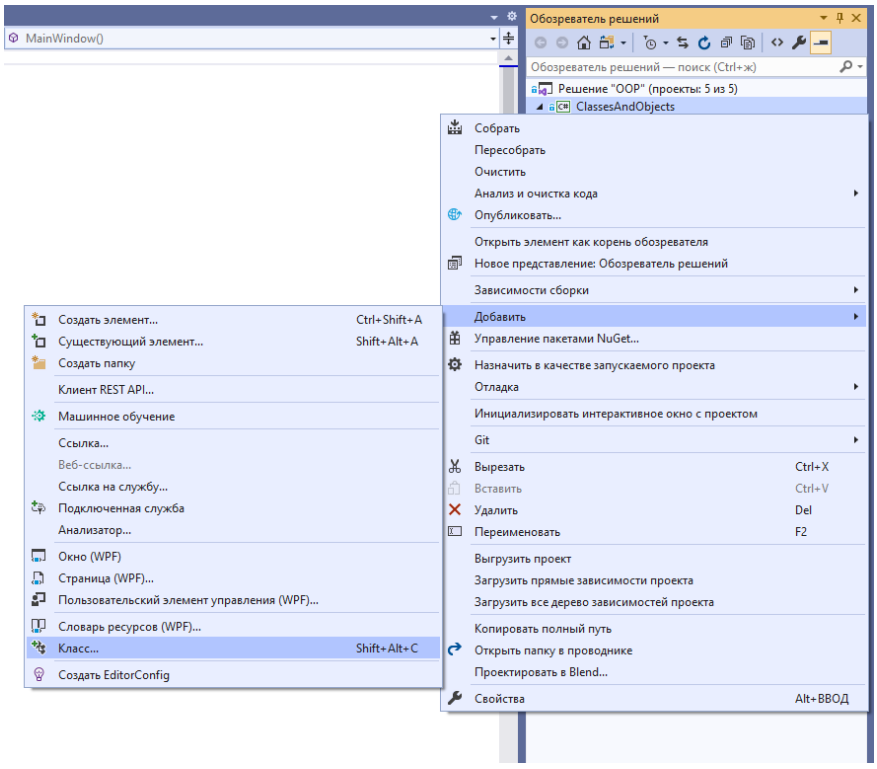
Описание и код NUnit доступны на официальном ресурсе: http://nunit.org

Создание простого набора тестов с использованием NUnit

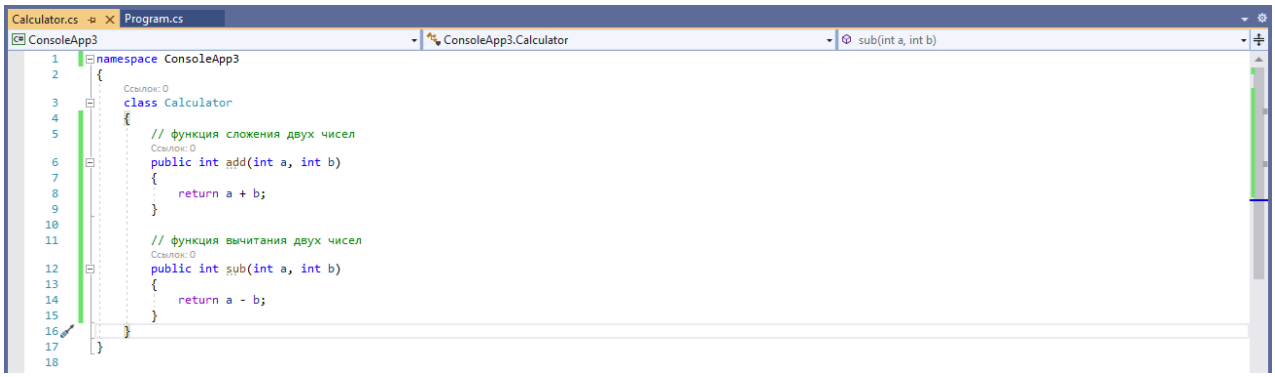
Создайте проект типа «Консольное приложение (.NET Framework)»:



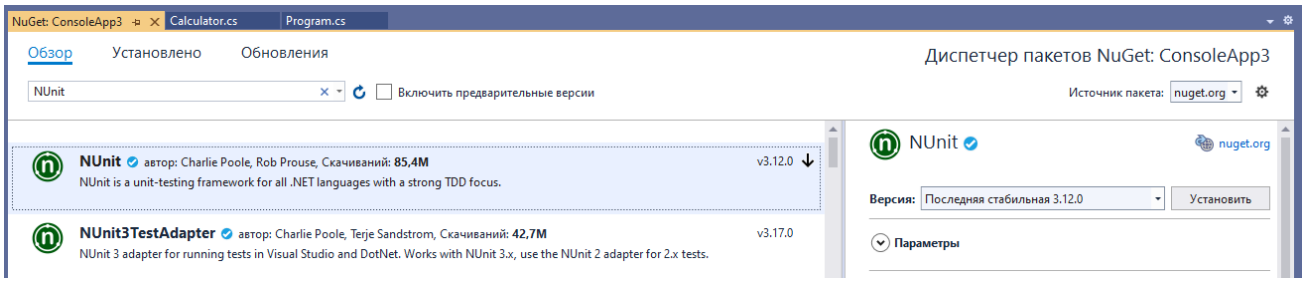
Добавьте в проект новый класс:



В создаваемом классе будут находиться функции для тестирования. Пример:



Подключение NUnit к проекту осуществляется через диспетчер пакетов NuGet. Для запуска тестов также необходимо установить NUnit3TestAdapter.

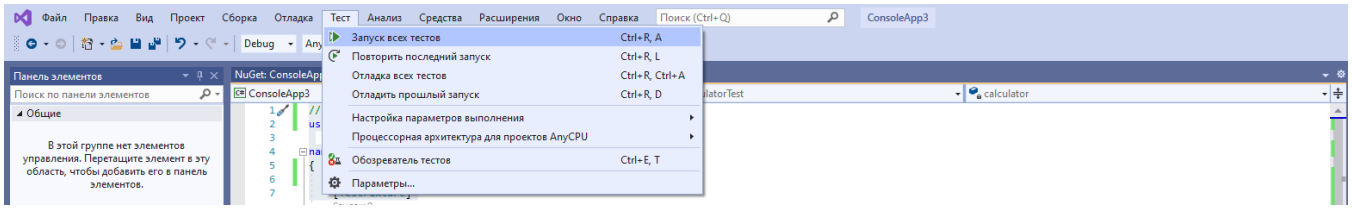


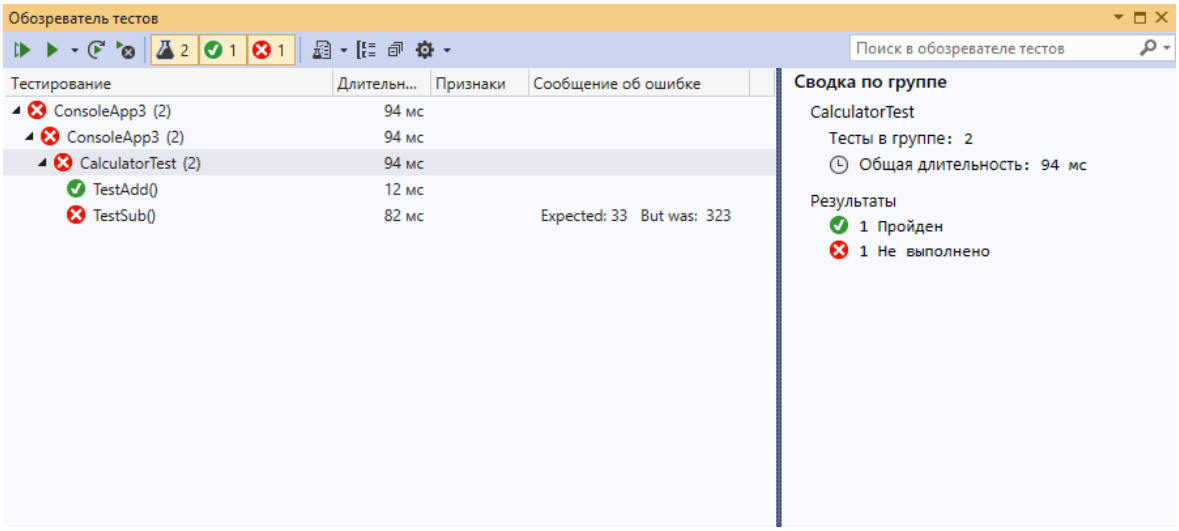
После этого добавьте тестовый класс. Как правило, название тестового класса содержит название тестируемого класса + постфикс Test.

В тестовом классе могут быть описаны тестовые сценарии. Тестовый сценарий - это вызов функции или последовательность вызовов функций и проверка возвращаемых значений. Если возвращаемые значения не соответствуют ожидаемым, тестовый сценарий считается не пройденным:

```
// подключение NUnit
using NUnit.Framework;

namespace ConsoleApp3
{
    // атрибут, указывающий на то, что это класс содержит тесты
    [TestFixture]
    class CalculatorTest
    {
        Calculator calculator = new Calculator();

        // атрибут, указывающий на то, что это тестовый метод
        [TestCase]
        public void TestAdd()
        {
            // тестирование функции сложения
            // если результат вызова функции не равен 33, тест не будет пройден
            Assert.AreEqual(33, calculator.add(3, 30));
        }

        [TestCase]
        public void TestSub()
        {
            // тестирование функции вычитания
            Assert.AreEqual(33, calculator.sub(326, 3));
        }
    }
}
```

После того, как описание тестовых сценариев будет завершено, запустить их можно через меню «Тест» -> «Запуск всех тестов», либо через обозреватель тестов («Тест» -> «Обозреватель тестов»):





Зелёным индикатором помечены успешно прошедшие тесты, красным индикатором – проваленные тесты.

Дополнительные функции NUnit

В процессе выполнения лабораторной работы могут быть использованы следующие функции тестирования:

Проверка функций на возникновение исключительных ситуаций

Предположим, существует функция вычисления остатка от деления:

```
public int mod(int a, int b)
{
    if (b <= 0) throw new ArgumentException("Делитель должен быть >= 0");
    return a % b;
}
```

В функции описано условие, которое может привести к возникновению исключительной ситуации. Чтобы протестировать корректность срабатывания условия, можно использовать тест следующего вида:

```
[TestCase]
public void TestMod()
{
    // получение исключения
    var exception = Assert.Throws<ArgumentException>(() => calculator.mod(2, 0));
    // сравнение полученного сообщения с ожидаемым
    Assert.That(exception.Message, Is.EqualTo("Делитель должен быть >= 0"));
    // проверка выполняется успешно, если исключение не было сгенерировано
    Assert.DoesNotThrow(() => calculator.mod(2, 1));
}
```

Проверка функций на возвращаемое состояние

Например, для функции проверки числа на чётность:

```
public bool isEven(int n)
{
    return n % 2 == 0;
}
```

Тест будет выглядеть следующим образом:

```
[TestCase]
public void TestIsEven()
{
    // проверка возвращаемого значения
    // в первом случае оно должно быть истинно, во втором ложно
    Assert.IsTrue(calculator.isEven(4));
    Assert.IsFalse(calculator.isEven(5));
}
```

Помимо истинности можно узнать, является ли объект пустым, неопределённым и т.д. Полный список Assert функций (с примерами) можно посмотреть в официальном репозитории NUnit https://docs.nunit.org/articles/nunit/writing-tests/assertions/assertion-models/classic.html

Задание:

1. Реализовать набор функций:
   - функция конвертации дюймов в сантиметры;
   - функция проверки числа на чётность;
   - функция выбора наибольшего значения из массива целых чисел;
   - функция вычисления остатка от деления.

   В случае возникновения ошибки, функции должны возвращать код ошибки.
2. Написать набор тестовых сценариев для реализованных тестовых функций. Тестовые сценарии должны охватывать все ветви выполнения функций.