Unit testing

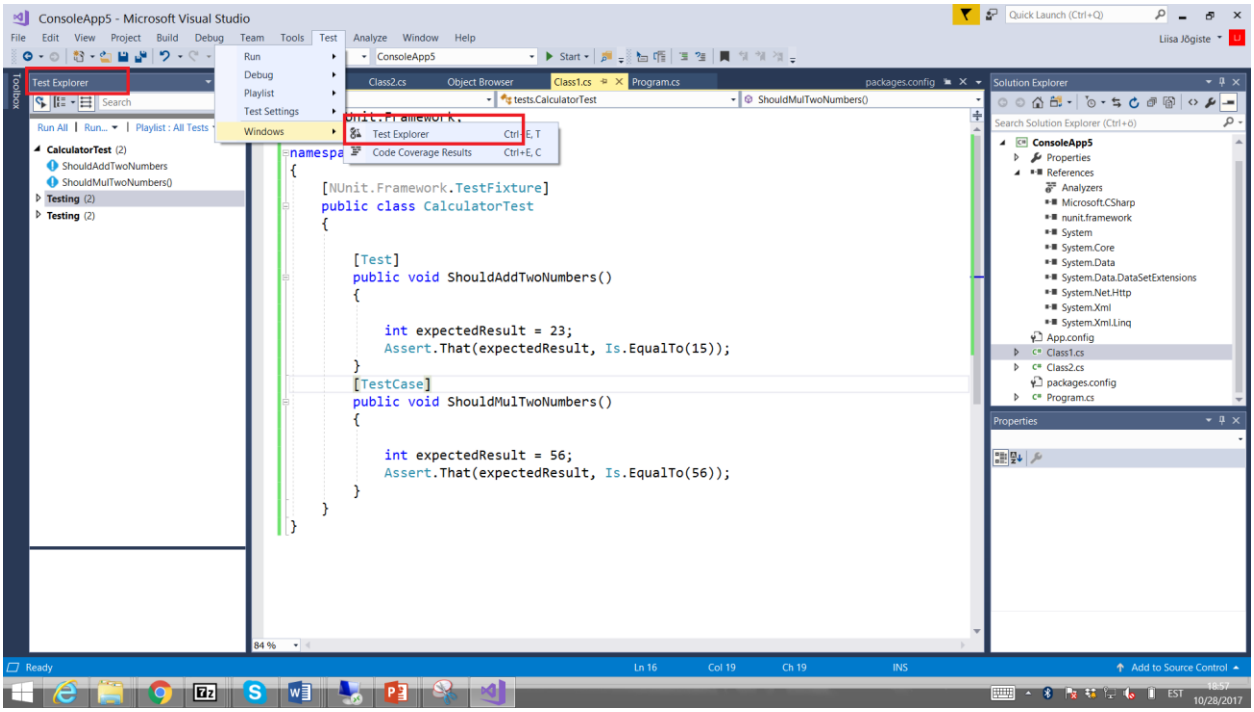
Adding N-Unit: watch: https://www.youtube.com/watch?v=f2NrKazjWes

Open visual studio. Then Tools->NuGet Package Manager -> Manage NuGet packages for this solution

Search and install:

- *Nunit*
- *NUnit 3 Test Adapter*

Opening tests window: Tests-> Windows -> Tests Explorer



Mac: Packages->Add pacage

Nunit syntax:

```
using NUnit.Framework;

namespace tests
{
    [TestFixture] //means "this is testing class"
    public class CalculatorTest
    {
        Calculator calc;

        [SetUp] //this method is called before each test
        public void TestSetup()
        {
           calc   = new Calculator();
```

```
        }
        [Test] //this means test
        public void ShouldAddTwoNumbers()
        {
            int expectedResult = calc.Add(7, 8);
            //Assert is used to test some sentences
            Assert.That(expectedResult, Is.EqualTo(15));
            Assert.AreEqual(expectedResult, 15);
        }
        [Test]
        public void ShouldMulTwoNumbers()
        {
            int expectedResult = calc.Mul(7, 8);
            //Assert is used to test some sentences
            Assert.That(expectedResult, Is.EqualTo(56));
            Assert.AreEqual(expectedResult, 56);
        }
    }
}
```

What can be tested:

- If values/objects are/are not equal
- If values are present/objects not null
- If certain exceptions are thrown
- If collections contain/do not contain certain values
- Etc

Example:

```
int[] array = new int[] { 1, 2, 3 };
Assert.That(array, Has.Exactly(1).EqualTo(3));
Assert.That(array, Has.Exactly(2).GreaterThan(1));
Assert.That(array, Has.Exactly(3).LessThan(100));


Assert.AreEqual(4, 2+2);
Assert.That(2+2, Is.EqualTo(4));
```

Exercise 1:

Create a class for calculator. The calculator has to work for both int and double, double is rounded by 2 numbers, example 8.76. NB, use method overloading!

Create a testing class. After completing each method test it!

The calculator has following methods:

- Subtract (*lahuta*) which takes two numbers as input and returns the result. Always the smaller number has to be subtracted from the bigger number!

- Multiply (*korruta*) which takes two numbers as input and returns the result
- Multiply (*korruta*) which takes three numbers as input and returns the result
- Square (*ruutu võtmine*) which takes one number as input and returns the result

The calculator has to store all the results of all the calculations made! Test the results also.

Exercise 2: first tests and then solve the task

Create a class called bus.

Add one method there:

- The method takes number of people and number of seats as parameter
- The method finds how many busses are needed and what is the number of people in the last bus

Example: *PutPeopleInBus(60,40) -> 2 busses, 20 people in last bus*

Create a separate test class and create tests for values:

- number of people 60, number of seats 40
- number of people 80, number of seats 40
- number of people 20, number of seats 40
- number of people 40, number of seats 40

Copy the test class content:

```
[TestFixture]
class BusTest
{
    Bus bus;
    [SetUp]
    public void TestSetup()
    {
        bus = new Bus(); //we create the object before test
    }

    [Test]
    public void Test60People_FindPeople()
    {
        bus.FindNumbers(60, 40);
        Assert.That(bus.GetNrOfPeople, Is.EqualTo(20));
    }

    [Test]
    public void Test60People_FindBusses()
    {
        bus.FindNumbers(60, 40);
        Assert.That(bus.GetNrOfBuses, Is.EqualTo(2));
    }

    [Test]
    public void Test80People_FindPeople()
    {
        bus.FindNumbers(80, 40);
```

```
    Assert.That(bus.GetNrOfPeople, Is.EqualTo(40));
}

[Test]
public void Test80People_FindBuses()
{
    bus.FindNumbers(80, 40);
    Assert.That(bus.GetNrOfBuses, Is.EqualTo(2));
}

[Test]
public void Test20People_FindPeople()
{
    bus.FindNumbers(20, 40);
    Assert.That(bus.GetNrOfPeople, Is.EqualTo(20));
}

[Test]
public void Test20People_FindBusses()
{
    bus.FindNumbers(20, 40);
    Assert.That(bus.GetNrOfBuses, Is.EqualTo(1));
}

[Test]
public void Test40People_FindPeople()
{
    bus.FindNumbers(40, 40);
    Assert.That(bus.GetNrOfPeople, Is.EqualTo(40));
}

[Test]
public void Test40People_FindBuses()
{
    bus.FindNumbers(40, 40);
    Assert.That(bus.GetNrOfBuses, Is.EqualTo(1));
}
```

Tip: we cannot have a method returning two values, thereby we should use class properties and create 2 methods for returning their values.

Exercise 3:

Create a class for generating car registration numbers in a format 000XXX (3 numbers, 3 letters).

In the class there should be a method that takes either 3 letters as an input and then creates 4 unique car regisration numbers containing the letters or numbers.

Example:

GenerateCodes(„AXT“) -> 345AXT, 456AXT, 783AXT, 987AXT

All generated codes are stored in a list.

Create a test file and test that:

- Codes are always generated 4 at once
- That codes contain the numbers/letters given in the method
- The codes are unique

Exercise 4:

The goal is to create a wallet containing money. Money has amount and currency. Wallet can contain different monies.

For example:
Wallet contains 12 euros and 14 us dollars.
Adding another 10 Euros gives a wallet with 22 Euros and 14 USD.
Adding 10 Estonian crowns gives a wallet with 22 EUR, 14 USD and 10 EEK.

Money has:
Constructor, Method for adding money, methods for returning amount and currency, method for subtracting money, method for comparing 2 money objects, method for finding money from wallet based on currency,

Wallet has:
Constructors, method for adding money to wallet, method for adding two wallets, method for subtracting money from wallet, method for comparing two wallets

Testing class:

```
using System;
using NUnit.Framework;
using System.Collections.Generic;

namespace Ex09
{

    [TestFixture]
    public class MoneyTest
    {
        private Money _12EUR;
        private Money _14EUR;
        private Money _7USD;
        private Money _21USD;

        private Wallet Wallet1;
        private Wallet Wallet2;

        [SetUp]
        protected void SetUp()
        {
```

```
    _12EUR = new Money(12,"EUR");
    _14EUR = new Money(14,"EUR");
    _7USD = new Money(7,"USD");
    _21USD = new Money(21,"USD");

    Wallet1 = new Wallet(_12EUR);
    Wallet2 = new Wallet(_14EUR);
}

[Test]
public void TestMoneyEquality()
{
    //Compare two money values

}

[Test]
public void SimpleAdd() //Test adding money
{
    // [12 EUR] + [14 EUR] == [26 EUR]

}

[Test]
public void MoneyMultiply()
{
    // [14 EUR] *2 == [28 EUR]

}

[Test]
public void MoneyNegate()
{
    // [14 EUR] negate == [-14 EUR]
}

[Test]
public void MoneySubtract()
{
    // [14 EUR] -[4 EUR] == [-10 EUR]
}

[Test]
public void PrintMoney()
{
    // [12 EUR] -> "12EUR"
}

[Test]
public void WalletContainsValue()
{
    //Create a wallet containing _12EUR
    //Find euros from wallet and see if the value is 12EUR
}

[Test]
public void MoneyWasAddedToWallet()
{
```

```
            // {[12 EUR]} + [14 EUR] == [26 EUR]
        }

        [Test]
        public void AreTwoWalletsEqual()
        {
            // {[12 EUR], [14EUR]}  ==  { [14EUR], [12EUR]}

        }

        [Test]
        public void WalletSimpleAdd() //Add money to Wallet
        {
            // {[12 EUR][7 USD]} + [14 EUR] == {[26 EUR][7 USD]}

        }


        [Test]
        public void WalletMultiply()
        {
            // {[12 EUR][7 USD]} *2 == {[24 EUR][14 USD]}

        }

        [Test]
        public void WalletNegate()
        {
            // {[12 EUR][7 USD]} negate == {[-12 EUR][-7 USD]}
        }

        [Test]
        public void WalletSumAdd()  //add 2 Wallets
        {
            // {[12 EUR][7 USD]} + {[14 EUR][21 USD]} == {[26 EUR][28 USD]}

        }




        [Test]
        public void SubtractMoneyFromWallet()
        {
            // [14 EUR][26USD]  - [10EUR] == [4 EUR][26USD]
        }

        [Test]
        public void SubtractWalletFromWallet()
        {
            // [14 EUR][26USD]  - [10EUR][6USD] == [4 EUR][20USD]
        }
    }
}
```