Oefeningen Unit Test met NUnit

1. Palindroom

Maak een class **Woord** in het project ***TDDCursusLibrary*** en vul volgende code aan:

```
public class Woord
{
    public string Text { get; set; }
    public Woord(string woord)
    {
        Text = woord;
    }
    public bool IsPalindroom()
    {
          throw new NotImplementedException();
    }
}
```

Je vult via de constructor een private variabele woord. Een method IsPalindroom() levert de waarde true op als het woord een palindroom is : een woord dat hetzelfde is als je het van voor naar achter leest en als je het van achter naar voor leest, bijvoorbeeld lepel.

Maak in het test project ***TDDCursusLibraryTest*** de class **WoordTest** de nodige tests voor de class Woord. Test de methode ***IsPalindroom***() met bv “lepel” (moet true teruggeven), “vork” (moet false teruggeven) en een lege string (moet true teruggeven)

Run de tests via de Test Explorer – deze moeten allemaal groen kleuren

2. Veiling

Je maakt met de voorgeschreven stappen van TDD een class die een veiling (verkoop) voorstelt en de bijbehorende unit test.

Gevraagd is om de volgende functionaliteit te programmeren in een class Veiling :

Je kan op een Veiling-object meerdere keren de **methode DoeBod** oproepen. Je geeft als parameter het bedrag van het bod mee.

Je kan op ieder moment de **readonly property HoogsteBod** oproepen. Deze methode geeft je het hoogst geboden bedrag terug.

Uit de analyse blijkt dat:

- Als nog geen enkel bod werd uitgevoerd, het hoogste bod gelijk is aan 0.
- Als een eerste bod werd uitgevoerd, het hoogste bod gelijk is aan het bedrag van het bod.
- Als meerdere keren een bod werd uitgevoerd, het hoogste bod gelijk is aan het bedrag van het hoogste bod.

Maak in het project ***TDDCursusLibrary*** de class Veiling **zonder** code (TDD principe):

```
public class Veiling
{
    private decimal hoogsteBod = 0m;
    public void DoeBod(decimal bedrag)
    {
         throw new NotImplementedException();

    }
    public decimal HoogsteBod
    {
        get
        {
            throw new NotImplementedException();
        }
    }
}
```

Maak nu eerst de class **VeilingTest** in het test project ***TDDCursusLibraryTest***:

Voorzie een [**SetUp**] om een nieuwe Veiling instatie aan te maken

Maak de test methoden aan (zie screenshot onderaan) en schrijf de test code

Run de tests, deze **falen** Aangezien de code in de class Veiling nog niet is ingevuld.

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

namespace TDDCursusLibrary.Test
{
    [TestFixture]
    0 references
    public class VeilingTest
    {
        private Veiling veiling;
        [SetUp]
        0 references
        public void Initialize()
        {
            veiling = new Veiling();
        }
        [Test]
        0 references
        public void HetHoogsteBodVanEenNieuweVeilingStaatOpNul()...
        [Test]
        0 references
        public void NaEenEersteBodIsHetHoogsteBodGelijkAanHetBedragVanDitBod()...
        [Test]
        0 references
        public void NaMeerdereBiedingenIsHetHoogsteBodGelijkAanHetBedragVanDitBod()...
    }
}
```

Vul nu de code aan in Veiling van TDDCursusLibrary totdat alle testen groen kleuren wanneer ze worden gerund.

3. ISBN

Je maakt met de voorgeschreven stappen van **TDD** een class die een ISBN-nummer voorstelt. Je maakt een bijhorende unit test.

De regels voor een ISBN-nummer zijn als volgt :

- Het bestaat uit 13 cijfers
- Een ISBN-nummer bevat een controlemechanisme :

  - Je maakt de som van de cijfers op de eerste 6 oneven posities

  - Je maakt de som van de cijfers op de eerste 6 even posities en je vermenigvuldigt deze som met 3

  - Je maakt de som van deze twee tussenresultaten

  - Je maakt het verschil van deze som en het naastgelegen hoger gelegen tiental

  - Het dertiende cijfer moet gelijk zijn aan dit verschil, tenzij het verschil 10 is, dan moet het dertiende cijfer gelijk zijn aan 0.

Voorbeeld : 9789027439642

som van de cijfers op de eerste 6 oneven posities = 9 + 8 + 0 + 7 + 3 + 6 = 33

som van de cijfers op de eerste 6 even posities = 7 + 9 + 2 + 4 + 9 + 4 = 35, dit x 3 = 105

som van deze twee tussenresultaten = 33 + 105 = 138

verschil van deze som en het naastgelegen hoger gelegen tiental = 140 – 138 = 2

dertiende cijfer moet gelijk zijn aan dit verschil : 2 = 2

```
public class Isbn
{
    public Isbn(long nummer)
    {
        throw new NotImplementedException();
    }
    public override string ToString()
    {
        throw new NotImplementedException();
    }
}
```

Maak (TDD) eerst de Test class **IsbnClass** in het project ***TDDCursusLibraryTest***

```
[TestFixture]
public class IsbnTest
{
    [Test]
    public void HetNummer0IsVerkeerd()
    {//vul de code aan…
    }
    [Test]
    public void EenNegatiefNummerIsVerkeerd()
    {//vul de code aan…
    }
    [Test]
```

```
        public void EenNummerMet12CijfersIsVerkeerd()
        {//vul de code aan…
        }
        [Test]
        public void EenNummerMet14CijfersIsVerkeerd()
        {//vul de code aan…
        }
        [Test]
        public void EenNummerMet13CijfersMetVerkeerdControleGetal2()
        {//vul de code aan…
        }
        [Test]
        public void EenNummerMet13CijfersMetCorrectControleGetal2()
        {//vul de code aan…
        }
        [Test]
        public void EenNummerMet13CijfersMetVerkeerdControleGetal0()
        {//vul de code aan…
        }
        [Test]
        public void EenNummerMet13CijfersMetCorrectControleGetal0()
        {//vul de code aan…
        }
    }
```

Run de tests, zij zullen eerst allemaal falen.

Pas dan de code in de **Isbn** class aan in het project ***TDDCursusLibrary*** totdat alle testen groen kleuren voorbeeld implementatie:

```
    public class Isbn
    {
        private const long GrootsteGetalMet13_Cijfers = 9999999999999L;
        private const long KleinsteGetalMet13_Cijfers = 1000000000000L;
        private long nummer;
        public Isbn(long nummer)
        {
            if (nummer < KleinsteGetalMet13_Cijfers ||
             nummer > GrootsteGetalMet13_Cijfers)
            {
                throw new ArgumentException();
            }
            var somEvenCijfers = 0L;
            var somOnEvenCijfers = 0L;
            var teVerwerkenCijfers = nummer / 10;
            for (int teller = 0; teller != 6; teller++)
            {
                somEvenCijfers += teVerwerkenCijfers % 10;
                teVerwerkenCijfers /= 10;
                somOnEvenCijfers += teVerwerkenCijfers % 10;
                teVerwerkenCijfers /= 10;
            }
            var controleGetal = somEvenCijfers * 3 + somOnEvenCijfers;
            var naastGelegenHoger10Tal = controleGetal -
             controleGetal % 10 + 10;
            var verschil = naastGelegenHoger10Tal - controleGetal;
            var laatsteCijfer = nummer % 10;
            if (verschil == 10)
            {
                if (laatsteCijfer != 0)
                {
                    throw new ArgumentException();
```

```
                }
            }
            else
            {
                if (laatsteCijfer != verschil)
                {
                    throw new ArgumentException();
                }
            }
            this.nummer = nummer;
        }
        public override string ToString()
        {
            return nummer.ToString();
        }
    }
```

4. Stub

Maak een class WinstService. Deze class heeft een dependency, uitgedrukt in een interface IOpbrengstDAO. Deze interface bevat één method-declaratie:

```
decimal TotaleOpbrengst();
```

De class WinstService heeft een tweede dependency, uitgedrukt in een interface IKostDAO. Deze interface bevat één method-declaratie:

```
decimal TotaleKost();
```

De class WinstService bevat een readonly-property Winst:

```
public decimal Winst;
```

Je berekent de winst door de totale kost af te trekken van de totale opbrengst.

Je schrijft de class **WinstService** en de bijbehorende unittest. In deze unittest gebruik je stubs voor **IOpbrengstDAO** en **IKostDAO**.

- In het project ***TDDCursusLibrary***:

De interface **IOpbrengstDAO** :

```
public interface IOpbrengstDAO
{
    decimal TotaleOpbrengst();
}
```

De interface **IKostDAO** :

```
public interface IKostDAO
{
    decimal TotaleKost();
}
```

De class **WinstService** :

```
public class WinstService
{
    private readonly IOpbrengstDAO opbrengstDAO;
    private readonly IKostDAO kostDAO;
    public WinstService(IOpbrengstDAO opbrengstDAO, IKostDAO kostDAO)
    {
        this.opbrengstDAO = opbrengstDAO;
        this.kostDAO = kostDAO;
    }
    public Decimal Winst
    {
        get
        {
            throw new NotImplementedException();
        }
    }
}
```

- In het project ***TDDCursusLibraryTest***:

De class **OpbrengstDAOStub** :

```
public class OpbrengstDAOStub : IOpbrengstDAO
{
    public decimal TotaleOpbrengst()
    {
        return 200m;
    }
}
```

De class **KostDAOStub** :

```
public class KostDAOStub : IKostDAO
{
    public decimal TotaleKost()
    {
        return 169m;
    }
}
```

De class **WinstServiceTest** :

```
[TestFixture]
public class WinstServiceTest
{
    private WinstService winstService;
    private IKostDAO kostDAO;
    private IOpbrengstDAO opbrengstDAO;
    [SetUp]
    public void Initialize()
    {
        kostDAO = new KostDAOStub();

        opbrengstDAO = new OpbrengstDAOStub();
        winstService = new WinstService(opbrengstDAO, kostDAO);
    }

    [Test]
    public void WinstIsOpbrengstMinKost()
    {
        //Vul de code aan
    }
}
```

5. Mock

Vervang de stubs in **WinstServiceTest** door mocks, aangemaakt met **Moq** (NuGet Package). Voeg ook de nodige verificaties toe.

```
    [TestFixture]
    public class WinstServiceTest
    {
        private WinstService winstService;
        private IKostDAO kostDAO;
        private IOpbrengstDAO opbrengstDAO;
        private /*in te vullen met Mock*/ mockKostDAO;
        private /*in te vullen met Mock*/ mockOpbrengstDAO;
        [SetUp]
        public void Initialize()
        {
            mockKostDAO = /*in te vullen met Mock*/
            kostDAO = mockKostDAO.Object;
            mockOpbrengstDAO = /*in te vullen met Mock*/
            opbrengstDAO = mockOpbrengstDAO.Object;
              //Trainen van mockKostDAO:
            mockKostDAO.Setup(eenKostDAO =>
             eenKostDAO.TotaleKost()).Returns(169m);
            //in te vullen Trainen van mockOpbrengstDAO: (geeft 200m terug)


            winstService = new WinstService(opbrengstDAO, kostDAO);
        }
        [Test]
        public void WinstIsOpbrengstMinKost()
        {
            Assert.AreEqual(31m, winstService.Winst);
            mockKostDAO.Verify(eenKostDAO => eenKostDAO.TotaleKost());
            /*in te vullen met Verificatie of TotaleOpbrengst() methode wordt
aangeroepen*/         }
    }
```