Oefening 2 – CharacterClass (NUnit)

1. **Voeg aan het project Business de volgende class toe**

```
namespace Business
{
    public enum Type
    {
        Elf,
        Ork
    }

    public class Character : INotifyPropertyChanged, IDisposable
    {
        private string _name;
        public Type Type { get; }

        public List<string> Weaponry { get; }

        public Character(Type type)
        {
            Type = type;
            Weaponry = new List<string>();
        }

        public Character(Type type, string name) : this(type)
        {
            Name = name;
        }

        public int Armor
        {
            get
            {
                switch (Type)
                {
                    case Type.Elf:
                        return 60;
                    case Type.Ork:
                        return 100;
                }
                throw new ArgumentOutOfRangeException();
            }
        }

        public bool IsDead => Health <= 0;

        public double Speed
        {
            get
            {
                switch (Type)
                {
                    case Type.Elf:
                        return 1.7;
                    case Type.Ork:
                        return 1.4;
                    default:
                        throw new ArgumentOutOfRangeException();
                }
            }
        }

        public int Wear { get; private set; } = 15;
        public int Health { get; private set; } = 100;

        public int Defense => Wear >= Armor ? 0 : Armor - Wear;

        public string Name
        {
            get { return _name; }
            set
            {
                _name = value;
                OnPropertyChanged("Name");
            }
        }

        public void Damage(int damage)
```

```
        {
            if (damage > 1000)
            {
                throw new ArgumentOutOfRangeException(nameof(damage));
            }
            Health -= damage - Defense;
        }

        public event PropertyChangedEventHandler PropertyChanged;

        public override string ToString()
        {
            return Name;
        }

        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }

        public void Dispose()
        {
        }
    }
}
```

2. **Voeg aan het project Business.Test een class** `CharacterTests`
3. **Duid deze class aan als Test class (welk attribuut gebruik je hiervoor?)**
4. **Voeg de volgende methode toe en duidt deze aan als test methode (welk attribuut gebruik je hiervoor ?)**
5. **Run de test. Welk resultaat krijg je? Wat wordt hier getest?**

```
public void ShouldSetName()
{
    const string expected = "John";
    Character c = new Character(Type.Elf, expected);

    Assert.That(c.Name, Is.EqualTo(expected));
    Assert.That(c.Name, Is.Not.Empty);
    Assert.That(c.Name, Contains.Substring("ohn"));
}
```

6. **Maak een nieuwe Test methode met naam ShouldSetNameCaseInsensitive()**

```
public void ShouldSetNameCaseInsensitive()
{
    const string expectedUpperCase = "JOHN";
    const string expectedLowerCase = "john";
    Character c = new Character(Type.Elf, expectedUpperCase);

    Assert.That(c.Name, Is.EqualTo(expectedLowerCase).IgnoreCase);
}
```

7. **Run de test. Welk resultaat krijg je? Wat wordt hier getest?**
8. **Verwijder de IgnoreCase uit de laatste lijn en run de test opnieuw. Welk resultaat krijg je nu ?**

9. **Maak een nieuwe Test methode:**

```
public void DefaultHealthIs100()
{
    Character c = new Character(Type.Elf);

    const int expectedHealth = 100;
    Assert.That(c.Health, Is.EqualTo(expectedHealth));
    Assert.That(c.Health, Is.Positive);
}
```

10. **Run de test. Wat is het resultaat?**
11. **Wijzig Is.Positive door Is.Negative en run de test opnieuw. Welk resultaat krijg je nu?**
12. **Voeg de 2 volgende test methoden toe:**

```
public void Elf_SpeedIsCorrect()
{
    Character c = new Character(Type.Elf);

    const double expectedHealth = 1.7;
    Assert.That(c.Speed, Is.EqualTo(expectedHealth));
}
public void Ork_SpeedIsCorrect()
{
    Character c = new Character(Type.Ork);

    const double expectedHealth = 1.4;
    Assert.That(c.Speed, Is.EqualTo(expectedHealth));
}
```

13. **Run deze tests. Wat is het resultaat?**
14. **Voeg de volgende test methode toe.**

```
public void Ork_SpeedIsCorrectWithTolerance()
{
    Character c = new Character(Type.Ork);

    const double expectedHealth = 0.3 + 1.1;
    Assert.That(c.Speed, Is.EqualTo(expectedHealth).Within(0.5));
    Assert.That(c.Speed, Is.EqualTo(expectedHealth).Within(1).Percent);

    //ranges of DateTimes
    var dt = new DateTime(2000, 1, 1);
    Assert.That(dt, Is.EqualTo(new DateTime(2001, 1, 1)).Within(TimeSpan.FromDays(366)));
    Assert.That(dt, Is.EqualTo(new DateTime(2001, 1, 1)).Within(366).Days);
}
```

15. **Run de test. Wat is het resultaat?**
16. **Verwijder de Within uit de lijnen van de laatste methode en run de test opnieuw. Wat is het resultaat ? Geef uitleg..**

17. **Voeg de volgende test methoden toe**

```
public void DefaultNameIsNull()
{
    Character c = new Character(Type.Elf);
    Assert.That(c.Name, Is.Null);
}
public void IsDead_KillCharacter_ReturnsTrue()
{
    Character c = new Character(Type.Elf);
    c.Damage(500);
    Assert.That(c.IsDead, Is.True);
}
```

18. **Run de tests. Wat wordt er getest?**
19. **Vervang de laatste lijn resp door de volgende :**

```
Assert.That(c.IsDead, Is.False);
Assert.IsTrue(c.IsDead);
Assert.IsFalse(c.IsDead);
```

20. **Run de laatste test opnieuw. Welke resultaten krijg je?**
21. **Voeg de volgende test methode toe**

```
public void CollectionTests()
{
    var c = new Character(Type.Elf);
    c.Weaponry.Add("Knife");
    c.Weaponry.Add("Pistol");

    Assert.That(c.Weaponry, Is.All.Not.Empty);
    Assert.That(c.Weaponry, Contains.Item("Knife"));
    Assert.That(c.Weaponry, Has.Exactly(2).Length);
    Assert.That(c.Weaponry, Has.Exactly(1).EndsWith("tol"));
    Assert.That(c.Weaponry, Is.Unique);
    Assert.That(c.Weaponry, Is.Ordered);

    var c2 = new Character(Type.Elf);
    c2.Weaponry.Add("Knife");
    c2.Weaponry.Add("Pistol");

    Assert.That(c.Weaponry, Is.EquivalentTo(c2.Weaponry));
}
```

22. **Run de test. Wat wordt er allemaal getest?**

23. **Voeg de volgende test methode toe**

```
public void SameCharacters_AreEqualByReference()
{
    Character c1 = new Character(Type.Elf);
    Character c2 = c1;

    Assert.That(c1, Is.SameAs(c2));
}
```

24. **Run de test. Wat is het resultaat? Leg uit.**
25. **Voeg de volgende test methode toe**

```
public void TestObjectOfCharacterType()
{
    object c = new Character(Type.Elf);

    Assert.That(c, Is.TypeOf<Character>());
}
```

26. **Run de test. Wat is het resultaat? Wijzig in de laatste lijn Character door DegreeConverter en run de test opnieuw. Wat is het resultaat?**
27. **Voeg de volgende test methode toe**

```
public void DefaultCharacterArmorShouldBeGreaterThan30AndLessThan100()
{
    Character c = new Character(Type.Elf);

   // Assert.That(c.Armor, Is.GreaterThan(30).And.LessThan(100));
    Assert.That(c.Armor, Is.InRange(30, 100));
}
```

28. **Zet de de voorlaatste lijn uit commentaar en run de test opnieuw. Resultaat ?:**
29. **Voeg de volgende test method toe en run de test:**

```
public void Damage_1000_ThrowsArgumentOutOfRange()
{
    var c = new Character(Type.Elf);

    Assert.Throws<ArgumentOutOfRangeException>(() => c.Damage(1001));
}
```

30. **Wijzig de laatste lijn en run de test opnieuw. Resultaat ?**

```
                Assert.That(() => c.Damage(1001),
Throws.TypeOf<ArgumentOutOfRangeException>());
```