Lab: Unit Testing

Problems for exercises and homework for the "C# OOP" course @ SoftUni".

Part I: Unit Testing Basics

1. Test Axe

Load provided solution in Visual Studio. Add new project **Tests**

Create a class **AxeTests**

Create the following tests:

- Test if weapon loses durability after each attack
- Test attacking with a broken weapon

Solution

```
[Test]
0 references
public void AxeLoosesDurabilityAfterAttack()
{
    Axe axe = new Axe(10, 10);
    Dummy dummy = new Dummy(10, 10);

    axe.Attack(dummy);

    Assert.That(axe.DurabilityPoints, Is.EqualTo(9), "Axe Durability doesn't change after attack.");
}
```

2. Test Dummy

Create a class **DummyTests**

Create the following tests:

- Dummy loses health if attacked
- Dead Dummy throws exception if attacked
- Dead Dummy can give XP
- Alive Dummy can't give XP

Hints

Follow the logic of the previous problem

3. Refactor Tests

Refactor the tests for **Axe** and **Dummy** classes

Make sure that:

- **Names** of test methods are **descriptive**
- You use **appropriate assertions** (assert equals vs assert true)
- You use **assertion messages**

- There are **no magic numbers**
- There is **no code duplication** (Don't Repeat Yourself)

Hints

Extract constants and private fields for **Axe** class

Create a method that executes **before each test**

Make use of constants and private fields, as well as add assertion messages

Follow the same logic for other test methods and **`TestDummy`** class

Part II: Dependencies

4. Fake Axe and Dummy

Test if hero gains XP when target dies

To do this, you need to:

- Make **Hero** class **testable** (use **Dependency Injection**)
- Introduce **Interfaces** for Axe and Dummy
    - Interface Weapon
    - Interface Target

Create fake Weapon and fake Dummy for the test

Hints

Create **IWeapon and ITarget** interface. Modify implementation methods to **make use of interfaces.** Modify both **Axe** and **Dummy** classes.

Use **Dependency Injection** for Hero class

Create **`HeroTests`** class and test gaining XP functionality by faking Weapon and Target classes

5. Mocking

Include **`Moq`** in the project dependencies, then:

- Mock fakes from previous problem Hints

Go to **`HeroTests`** and refactor the code, making use of **`Moq`**