# C# OOP Exam

# Heroes

## Overview

You have to create a basic **RPG game**. In the game, there are **maps**, **heroes**, and **weapons**.

## Setup

* Upload **only the** Heroesproject in every problem **except** **Unit Tests.**
* **Do not modify the interfaces or their namespaces.**
* Use **strong cohesion** and **loose coupling.**
* **Use inheritance and the provided interfaces wherever possible.**
  + This includes **constructors**, **method parameters,** and **return types.**
* **Do not** violate your **interface** **implementations** by adding **more public methods** or **properties** in the concrete class than the interface has defined.
* Make sure you have **no public fields** anywhere.
* **Exception messages** and **output messages** can be found in the **"Utilities"** folder.
* To solve this problem use **Visual Studio 2019,** and **netcoreapp 3.1.**

## Task 1: Structure (50 points)

**For this task's evaluation logic in the methods isn't included.**

You are given interfaces, and you have to implement their functionality in the **correct classes**.

There are **3** types of entities in the application: **Weapon**, **Hero,** and **Map**. There should also be **HeroRepository** and **WeaponRepository**.

### Weapon

A weapon is a **base class** of any **type of weapon** and it **should not be able to be instantiated**.

#### Data

* **Name** – string
  + If the name **is null or whitespace,** throw an **ArgumentException** with the message: **"Weapon type cannot be null or empty."**
  + All names are unique.
* **Durability** – int
  + If the durability is below 0**,** throw an **ArgumentException** with the message: **"Durability cannot be below 0."**

#### Behavior

##### abstract int DoDamage()

The **DoDamage()** method returns the damage of each weapon:

* **Mace** - 25 damage
* **Claymore** - 20 damage

Whenever the **DoDamage** method is invoked, the weapon's durability is decreased by 1. If the **weapon's durability becomes 0**, the method should **return 0**.

#### Constructor

The constructor of the **Weapon** class should accept the following parameters:

**string name, int durability**

#### Child Classes

There are two concrete types of **Weapon**:

##### Mace

The **Mace** does 25 damage.

The mace takes the following values upon initialization:

**string name, int durability**

##### Claymore

The **Claymore** does 20 damage.

The claymore takes the following values upon initialization:

**string name, int durability**

### Hero

Hero is a **base class** for any **type of hero** and it **should not be able to be instantiated**.

#### Data

* **Name** - **string**
  + If the name **is null or whitespace,** throw an **ArgumentException** with the message: **"Hero name cannot be null or empty."**
  + All names are unique
* **Health** - **int**
  + If the health is below 0**,** throw an **ArgumentException** with the message: **"Hero health cannot be below 0."**
* **Armour** - **int**
  + If the armour is below 0**,** throw an **ArgumentException** with the message: **"Hero armour cannot be below 0."**
* **IsAlive** - **bool**
  + Calculated property, if the health is above zero, returns true.
* **IWeapon** - **Weapon**
  + If the weapon is null**,** throw an **ArgumentException** with the message: **"Weapon cannot be null."**

#### Behavior

#### void AddWeapon(IWeapon weapon)

This method adds a weapon to the given hero. A hero can have only **one** weapon.

##### void TakeDamage(int points)

The **TakeDamage()** method decreases the **Hero**'s health. First, the **armour is reduced**. If the goes below or **reaches** zero set the armour to zero and transfer the damage to health points. If the health points are less than or equal to zero, set the health to zero, the hero is dead.

**Constructor**

The constructor of the **Hero** class should accept the following parameters:

### string name, int health, int armour

#### Child Classes

There are two types of **Hero**:

##### Barbarian

The constructorshould take the following values upon initialization:

### string name, int health, int armour

**Knight**

The constructorshould take the following values upon initialization:

### string name, int health, int armour

### Map

#### Behavior

##### string Fight(ICollection<IHero> heroes)

Separates all heroes into two types - **knights** and **barbarians**. The battle continues until one of the teams is completely dead (**all heroes have 0 health**). The **knights** attack **first** and **after** that the **barbarians**.

The attack happens like so:

* Each **knight** (**if he is alive**) **attacks** each **barbarian** **once** and **inflicts** damage **equal** to the **damage of his weapon**.
* Next, each **barbarian** (**if he is alive**) attacks each knight and **inflicts** damage **equal** to the **damage of his weapon**.

The method **returns a string** with the winning team:

* When **knights win**, print: **"The knights took { number of death knights } casualties but won the battle."**
* When **barbarians win**, print: **"The barbarians took { number of death barbarians } casualties but won the battle."**

**HeroRepository**

A **repository** for the **heroes**.

#### Data

* Models - **a** **collection of heroes (unmodifiable)**

#### Behavior

##### void Add(IHero model)

* **Adds a** **hero** to the **collection**.

**bool Remove(IHero model)**

* **Removes a** hero from the **collection**. **Returns true** if the deletion was **successful**, **otherwise** - **false**.

**IHero FindByName(string name)**

* **Returns** the **first hero with the given name**. **Otherwise**, returns **null**.

### WeaponRepository

A **repository** for **weapons**.

#### Data

* Models - **a** **collection of weapons (unmodifiable)**

#### Behavior

##### void Add(IWeapon model)

* **Adds a weapon** to the **collection**.

**bool Remove(IWeapon model)**

* **Removes** a weapon from the **collection**. **Returns true** if the deletion was **successful**, **otherwise** - **false**.

**IWeapon FindByName(string name)**

* **Returns** the **first weapon with the given name**. **Otherwise**, returns **null**.

## Task 2: Business Logic (150 points)

### The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces, which you have to implement in the correct classes.

**Note: The** Controller **class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

The first interface is **I**Controller. Your task is to create a Controllerclass, which implements the interface and implements all of its methods. The constructor of Controllerdoes not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the Controllerclass, go into the **Engine** class constructor and uncomment the "**this.controller = new Controller();**" line.

### Data

You need to keep track of some things, this is why you need some private fields in your controller class:

* **heroes** - **HeroRepository**
* **weapons** - **WeaponRepository**

### Commands

There are several **commands**, which control the **business** **logic** of the **application**. They are **stated** **below**. The **command name** passed to the methods will **always** be **valid**!

#### CreateHero Command

##### Parameters

* type - string
* name - string
* health - int
* armour - int

##### Functionality

Creates a **hero** with the given parameters and adds it to the **HeroRepository**.

* If a **hero with the given name exists**, throwan **InvalidOperationException** with **the following message:** **"The hero { name } already exists."**
* If the **hero type is invalid**, throwan **InvalidOperationException** with **the following message:** **"Invalid hero type."**
* If the **Hero** is **added successfully to the repository**, **return** the following **message**:
  + For a **knight** print: **"Successfully added Sir { name } to the collection."**
  + For a **barbarian** print: **"Successfully added Barbarian { name } to the collection."**

#### CreateWeapon Command

##### Parameters

* **type** - **string**
* **name** - **string**
* **durability** - **int**

##### Functionality

Creates a **weapon** with the given parameters and **adds** it to the **WeaponRepository**. **Valid weapon** types are: "**Claymore**" and "**Mace**":

* If a weapon with the given **name** exists, throw an **InvalidOperationException** with the message: **"The weapon { name } already exists."**
* If the weapon **type** is **invalid**, throw an **InvalidOperationException** with the message: **"Invalid weapon type."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"A { weapon type } { weapon name } is added to the collection.".** Keep in mind that the weapon **type** should be **lowercase**.

#### AddWeaponToHero Command

##### Parameters

* weaponName - string
* heroName - string

##### Functionality

**Adds** a weapon with the given **name**, to a **hero** with the given **name**.If the operation is **successful** the weapon should be **removed** from the repository.

* If a **hero** with the given **name does not exist**, **throw an InvalidOperationException** with **the following message**: **"Hero { name of the hero } does not exist."**
* If a **weapon** with the given **name does not exist**, **throw an InvalidOperationException** with **the following message**: **"Weapon { name of the weapon } does not exist."**
* If the **hero already has a weapon**, **throw an InvalidOperationException** with **the following message**: **"Hero { name of the hero } is well-armed."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"Hero {name of the hero} can participate in battle using a { weapon type }."** Keep in mind that the weapon **type** should be **lowercase**.

#### StartBattle Command

##### Functionality

A **map** is created and a **fight** starts with all the heroes who are **alive** and **have weapons**! Returns the result from the **Fight()** method.

#### HeroReport Command

##### Functionality

Returns information about each hero separated with a new line. Order them by **hero type alphabetically**, then by **health descending**, then **by hero name alphabetically**:

**"{ hero type }: { hero name }  
--Health: { hero health }  
--Armour: { hero armour }**

**--Weapon: { weapon name }/Unarmed**

**{ hero type }: { hero name }  
--Health: { hero health }  
--Armour: { hero armour }**

**--Weapon: { weapon }/Unarmed**

**(…)"**

**Note: Do not use** "**\n\r**" **for a new line. There is not an empty row between different heroes.**

#### Exit Command

##### Functionality

Ends the program.

### Input / Output

You are provided with one interface, which will help you with the correct execution process of your program. The interface is IEngine and the class implementing this interface should read the input and when the program finishes, this class should print the output.

You are given the **Engine** class with written logic in it. For the code to be **compiled**, some parts are **commented on**, **don’t forget to uncomment them**.

#### Input

Below, you can see the **format** in which **each command** will be given in the input:

* **CreateHero { type } { name } { health } { armour }**
* **CreateWeapon { type } { name } { durability }**
* **AddWeaponToHero { weaponName } { heroName }**
* **StartBattle**
* **HeroReport**
* **Exit**

#### Output

Print the output from each command when issued. If an exception is thrown during any of the commands' execution, print the exception message.

#### Examples

|  |
| --- |
| **Input** |
| **CreateHero Knight Wilmetta 50 90**  **CreateHero Knight Dave 60 95**  **CreateHero Knight Serlon 40 100**  **CreateHero Knight ldis 80 110**  **CreateHero Barbarian Alessia 110 40**  **CreateHero Barbarian Casey 85 25**  **CreateHero Barbarian Emersyn 100 35**  **CreateHero Barbarian Attila 115 47**  **CreateHero Barbarian Berenger 120 15**  **CreateWeapon Claymore Almace 10**  **CreateWeapon Claymore Caliburn 12**  **CreateWeapon Claymore Dainsleif 9**  **CreateWeapon Claymore Durandal 13**  **CreateWeapon Claymore Dyrnwyn 7**  **CreateWeapon Mace Narcoleptic 15**  **CreateWeapon Mace Stormcaller 14**  **CreateWeapon Mace Willbreaker 11**  **CreateWeapon Mace Justifier 9**  **AddWeaponToHero Almace Serlon**  **AddWeaponToHero Durandal ldis**  **AddWeaponToHero Dyrnwyn Alessia**  **AddWeaponToHero Narcoleptic Casey**  **AddWeaponToHero Stormcaller Emersyn**  **AddWeaponToHero Willbreaker Attila**  **AddWeaponToHero Justifier Berenger**  **AddWeaponToHero Dainsleif Dave**  **AddWeaponToHero Caliburn Wilmetta**  **StartBattle**  **Exit** |
| **Output** |
| **Successfully added Sir Wilmetta to the collection.**  **Successfully added Sir Dave to the collection.**  **Successfully added Sir Serlon to the collection.**  **Successfully added Sir ldis to the collection.**  **Successfully added Barbarian Alessia to the collection.**  **Successfully added Barbarian Casey to the collection.**  **Successfully added Barbarian Emersyn to the collection.**  **Successfully added Barbarian Attila to the collection.**  **Successfully added Barbarian Berenger to the collection.**  **A claymore Almace is added to the collection.**  **A claymore Caliburn is added to the collection.**  **A claymore Dainsleif is added to the collection.**  **A claymore Durandal is added to the collection.**  **A claymore Dyrnwyn is added to the collection.**  **A mace Narcoleptic is added to the collection.**  **A mace Stormcaller is added to the collection.**  **A mace Willbreaker is added to the collection.**  **A mace Justifier is added to the collection.**  **Hero Serlon can participate in battle using a claymore.**  **Hero ldis can participate in battle using a claymore.**  **Hero Alessia can participate in battle using a claymore.**  **Hero Casey can participate in battle using a mace.**  **Hero Emersyn can participate in battle using a mace.**  **Hero Attila can participate in battle using a mace.**  **Hero Berenger can participate in battle using a mace.**  **Hero Dave can participate in battle using a claymore.**  **Hero Wilmetta can participate in battle using a claymore.**  **The knights took 2 casualties but won the battle.** |

|  |
| --- |
| **Input** |
| **CreateHero Knight 50 90**  **CreateHero Knight Dave 60 -5**  **CreateHero Knight Serlon -5 100**  **CreateHero Knight ldis 80 110**  **CreateHero Barbarian Alessia 110 40**  **CreateWeapon Claymore 10**  **CreateWeapon Claymore Caliburn -5**  **CreateWeapon Claymore Dainsleif 9**  **CreateWeapon Mace Willbreaker 11**  **CreateWeapon Mace Stormcaller 12**  **AddWeaponToHero Dainsleif ldis**  **AddWeaponToHero Willbreaker Alessia**  **AddWeaponToHero Willbreaker Alessia**  **AddWeaponToHero Stormcaller Alessia**  **CreateHero Knight Berenger 50 90**  **StartBattle**  **HeroReport**  **Exit** |
| **Output** |
| **Hero name cannot be null or empty.**  **Hero armour cannot be below 0.**  **Hero health cannot be below 0.**  **Successfully added Sir ldis to the collection.**  **Successfully added Barbarian Alessia to the collection.**  **Weapon type cannot be null or empty.**  **Durability cannot be below 0.**  **A claymore Dainsleif is added to the collection.**  **A mace Willbreaker is added to the collection.**  **A mace Stormcaller is added to the collection.**  **Hero ldis can participate in battle using a claymore.**  **Hero Alessia can participate in battle using a mace.**  **Weapon Willbreaker does not exist.**  **Hero Alessia is well-armed.**  **Successfully added Sir Berenger to the collection.**  **The knights took 0 casualties but won the battle.**  **Barbarian: Alessia**  **--Health: 0**  **--Armour: 0**  **--Weapon: Willbreaker**  **Knight: Berenger**  **--Health: 50**  **--Armour: 90**  **--Weapon: Unarmed**  **Knight: ldis**  **--Health: 15**  **--Armour: 0**  **--Weapon: Dainsleif** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with **Car** and **Garage** classes inside. The **Garage** class has some methods, fields, and one constructor, which are working properly. The  **Car** class has three properties and a constructor. You are **NOT ALLOWED** to change any class. Cover the whole **Garage** class with unit tests to make sure that the class is working as intended.

You are provided with a **unit test project** in the **project skeleton**.

Do **NOT** use **Mocking** in your unit tests!