## C# OOP Exam - 12 April 2020

## Overview

In this exam your task will be to create a basic Shooter game. In the game there are **Map**, **Players** of different teams and **Guns**.

## Setup

* Upload **only the** CounterStrikerproject 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

## Task 1: Structure (50 points)

Evaluation logic in the methods isn't included for this task.

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

There are **3** types of entities in the application: **Gun**, **Player**, **Map**. There should also be **GunRepository** and **PlayerRepository**.

### Gun

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

#### Data

* **Name** – string
  + If the name **is null or whitespace,** throw an **ArgumentException** with message: "Gun cannot be null or empty."
  + All names are unique
* **BulletsCount** – int
  + If the bullets count are below 0**,** throw an **ArgumentException** with message: "Bullets cannot be below 0."

#### Behavior

##### abstract int Fire()

The **Fire()** method returns the number of bullets fired. **Pistol** can fire only 1 bullet and the **Rifle** only 10 at once, not more, not less. If no bullets are fired, the method should return 0.

#### Constructor

A **Gun** should take the following values upon initialization:

string name, int bulletsCount

#### Child Classes

There are two types of **Gun**:

##### Pistol

Constructorshould take string name and int bulletsCount upon initialization.

Pistol can strike 1 bullet at a time.

##### Rifle

Constructorshould take string name and int bulletsCount upon initialization.

Rifle can strike 10 bullets at a time.

### Player

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

#### Data

* **Username** - **string**
  + If the username **is null or whitespace,** throw an **ArgumentException** with message: "Username cannot be null or empty."
  + All names are unique
* **Health** - **int**
  + If the health is below 0**,** throw an **ArgumentException** with message: "Player health cannot be below or equal to 0."
* **Armor** - **int**
  + If the armor is below 0**,** throw an **ArgumentException** with message: "Player armor cannot be below 0."
* **IsAlive** - **bool**
  + If the health is above zero
* **Gun** - **Gun**
  + If the gun is null**,** throw an **ArgumentException** with message:

"Gun cannot be null or empty."

#### Behavior

##### void TakeDamage(int points)

The **TakeDamage()** method decreases the **Player**'s health. First you need to reduce the armor. If the armor reaches 0, transfer the damage to health points. If the health points are less than or equal to zero, the player is dead.

#### Constructor

A **Player** should take the following values upon initialization:

string username, int health, int armor, IGun gun

#### Child Classes

There are two types of **Player**:

##### Terrorist

Constructorshould take the following values upon initialization:

string username, int health, int armor, IGun gun

##### CounterTerrorist

Constructorshould take the following values upon initialization:

string username, int heаlth, int armor, IGun gun

### Map

#### Behavior

##### string Start(ICollection<IPlayer> players)

Separates the players in two types - **Terrorist** and **Counter Terrorist**. The game continues until one of the teams is completely dead (**all players have 0 health**). The **terrorists** attack **first** and **after** that the **counter terrorists**. The attack happens like that: Each terrorist (**if he is alive**) **shoots** on each counter terrorist **once** and **inflicts** damage **equal** to the **bullets fired** and after that each counter terrorist (**if he is alive**) shoots on each terrorist.

Return a string that says which of the teams won:

* **"Counter Terrorist wins!"**
* **"Terrorist wins!"**

### GunRepository

The **gun repository** is a **repository** for all **guns** in the game.

#### Data

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

#### Behavior

##### void Add(IGun gun)

* If the gun is null**,** throw an **ArgumentException** with message: "Cannot add null in Gun Repository".
* **Adds** a **gun** in the **collection**.

##### bool Remove(IGun gun)

* **Removes** a **gun** from the **collection**. **Returns true** if the removal was **sucessful**, **otherwise** - **false**.

**IGun FindByName(string name)**

* **Returns** the **first** **gun** with the **given name**, if there is such gun. **Otherwise**, returns **null**.

### PlayerRepository

The **player repository** is a **repository** for all **players** in the game.

#### Data

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

#### Behavior

##### void Add(IPlayer player)

* If the player is null**,** throw an **ArgumentException** with message: "Cannot add null in Player Repository".
* **Adds** a **player** in the **collection**.

##### bool Remove(IPlayer player)

* **Removes** a **player** from the **collection**. **Returns true** if the removal was **sucessful**, **otherwise** - **false**.

**IPlayer FindByName(string name)**

* **Returns** the **first** **player** with the **given username**, if there is such player. **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. You must create a Controllerclass, which implements the interface and 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.

### Data

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

* **guns** - **GunRepository**
* **players** – **PlayerRepository**
* **map - IMap**

### Commands

There are several **commands**, which control the **business** **logic** of the **application**. They are **stated** **below**.

#### AddGun Command

##### Parameters

* type - string
* name - string
* bulletsCount - int

##### Functionality

**Adds** a **Gun** and **adds** it to the **GunRepository**. **Valid** types are: "**Pistol**" and "**Rifle**".

If the **Gun** **type** is **invalid**, you have to **throw an ArgumentException** with **the following message:**

* "Invalid gun type."

If the **Gun** is **added successfully**, the method should **return** the following **string**:

* "Successfully added gun {gunName}."

#### AddPlayer Command

##### Parameters

* type - string
* username – string
* health – int
* armor – int
* gunName - string

##### Functionality

**Creates** a **Player** of the **given type** and **adds** it to the **PlayerRepository**. **Valid** types are: "**Terrorist**" and "**CounterTerrorist**".

If the **gun** is **not found** throw **ArgumentException** with message:

* "Gun cannot be found!"

If the player **type** is **invalid**, throw an **ArgumentException** with message:

* "Invalid player type!"

The **method** should **return** the following **string** if the **operation** is **successful**:

* "Successfully added player {playerUsername}."

#### StartGame Command

##### Functionality

Game starts with all players that are **alive**! Returns the result from the **Start()** method.

#### Report Command

##### Functionality

Returns information about each player separated with a new line. Order them by type alphabetically, then by health descending, then by username alphabetically. You can use the overridden **ToString Player** method.

"{player type}: {player username}");

"--Health: {player health}");

"--Armor: {player armor}");

"--Gun: {player gun name}")

**Note: Use \r\n or Environment.NewLine for a new line and don't forget to trim the end if you use StringBuilder.**

#### 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. In order for the code to be **compiled**, some parts are **commented**, **don't forget to uncomment them**. The **try-catch block** is also **commented** in order for the program to **throw exceptions and for you to see them**, **uncomment it** when you are **ready** with this too.

#### Input

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

* **AddGun** **{type} {name} {bulletsCount}**
* **AddPlayer {type} {username} {health} {armor} {gunName}**
* **StartGame**
* **Report**
* **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** |
| AddGun Rifle Express 100  AddGun Rifle Buffalo 100  AddGun Rifle Assault 100  AddGun Granate Invalid 100  AddGun Pistol Kolibri 5  AddGun Pistol Makarov 15  AddGun Pistol Magnum 3  AddGun Pistol 3  AddPlayer Terrorist Shopoff 50 50 Express  AddPlayer Terrorist Kris 50 50 Buffalo  AddPlayer Terrorist 50 50 Express  AddPlayer Terrorist Atanas 50 50 Invalid  AddPlayer Terrorist Atanas -10 50 Express  AddPlayer Terrorist Atanas 20 -50 Express  AddPlayer CounterTerrorist John 50 50 Kolibri  AddPlayer CounterTerrorist Peter 30 30 Makarov  AddPlayer Player Invalid 30 30 Makarov  StartGame  Report  Exit |
| **Output** |
| Successfully added gun Express.  Successfully added gun Buffalo.  Successfully added gun Assault.  Invalid gun type!  Successfully added gun Kolibri.  Successfully added gun Makarov.  Successfully added gun Magnum.  Gun cannot be null or empty.  Successfully added player Shopoff.  Successfully added player Kris.  Username cannot be null or empty.  Gun cannot be found!  Player health cannot be below or equal to 0.  Player armor cannot be below 0.  Successfully added player John.  Successfully added player Peter.  Invalid player type!  Terrorist wins!  CounterTerrorist: John  --Health: 0  --Armor: 0  --Gun: Kolibri  CounterTerrorist: Peter  --Health: 0  --Armor: 0  --Gun: Makarov  Terrorist: Kris  --Health: 50  --Armor: 46  --Gun: Buffalo  Terrorist: Shopoff  --Health: 50  --Armor: 45  --Gun: Express |

|  |
| --- |
| **Input** |
| AddGun Rifle Express 1000  AddGun Rifle Buffalo 1000  AddGun Rifle Assault 1000  AddGun Pistol Kolibri 20  AddGun Pistol Makarov 20  AddGun Pistol Magnum 20  AddPlayer Terrorist Shopoff 50 44 Makarov  AddPlayer Terrorist Kris 50 0 Magnum  AddPlayer Terrorist Atanas 50 10 Kolibri  AddPlayer CounterTerrorist John 100 100 Express  AddPlayer CounterTerrorist Peter 100 100 Buffalo  StartGame  Report  Exit |
| **Output** |
| Successfully added gun Express.  Successfully added gun Buffalo.  Successfully added gun Assault.  Successfully added gun Kolibri.  Successfully added gun Makarov.  Successfully added gun Magnum.  Successfully added player Shopoff.  Successfully added player Kris.  Successfully added player Atanas.  Successfully added player John.  Successfully added player Peter.  Counter Terrorist wins!  CounterTerrorist: John  --Health: 100  --Armor: 89  --Gun: Express  CounterTerrorist: Peter  --Health: 100  --Armor: 89  --Gun: Buffalo  Terrorist: Atanas  --Health: 0  --Armor: 0  --Gun: Kolibri  Terrorist: Kris  --Health: 0  --Armor: 0  --Gun: Magnum  Terrorist: Shopoff  --Health: 0  --Armor: 0  --Gun: Makarov |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with **Robot** and **RobotManager** classes inside. The class will have some methods, fields and one constructor, which are working properly. You are **NOT ALLOWED** to change any classes. Cover the whole 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!