# C# OOP Regular Exam

## Overview

You have to create a **Gym** project, which keeps track of the athletes and gym equipment. There will be different types of gyms, suitable for a particular athlete.

## Setup

* Upload **only the** Gymproject 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)

**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: **Athlete, Equipment, and Gym**. There should also be **EquipmentRepository**.

### Athlete

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

#### Data

* **FullName** - **string**
  + If the full name **is null or empty,** throw an **ArgumentException** with the message: **"Athlete name cannot be null or empty."**
  + All names are unique
* **Motivation** - **string**
  + If the motivation **is null or empty,** throw an **ArgumentException** with the message: **"The motivation cannot be null or empty."**
* **Stamina** - **int**
  + The stamina of an **athlete**
* **NumberOfMedals** - **int**
  + The number of medals that an athlete has earned
  + If the number of medals is below **0,** throw an **ArgumentException** with the message:

**"Athlete's number of medals cannot be below 0."**

#### Behavior

##### abstract void Exercise()

The **Exercise()** method increases the Athlete’s stamina.

#### Constructor

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

**string fullName, string motivation,** **int numberOfMedals, int stamina**

#### Child Classes

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

##### Boxer

Has an **initial stamina of 60**.

**Can train only in a BoxingGym.**

The constructorshould take the following values upon initialization:

**string fullName, string motivation, int numberOfMedals**

###### **Behavior**

**void Exercise()**

* The method **increases** the boxer’s stamina by 15.
  + If total stamina **exceeds 100,** set the stamina to 100 and throw an **ArgumentException** with the message: **"Stamina cannot exceed 100 points."**

##### Weightlifter

Has an **initial stamina of 50**.

**Can train only in a WeightliftingGym.**

The constructorshould take the following values upon initialization:

**string fullName, string motivation, int numberOfMedals**

**Behavior**

**void Exercise()**

* The method **increases** the weightlifter’s stamina by 10.
  + If total stamina **exceeds 100,** set the stamina to 100 and throw an **ArgumentException** with the message: **"Stamina cannot exceed 100 points."**

### Equipment

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

#### Data

* **Weight** - **double**
* **Price** - **decimal**

#### Constructor

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

**double weight, decimal price**

#### Child Classes

There are two concrete types of Equipment:

##### BoxingGloves

**Weights** 227 grams and the **price** of **120**.

The Constructorshould take no values upon initialization.

##### Kettlebell

**Weights** 10000 grams and **price** of **80**.

The constructorshould take no values upon initialization.

### Gym

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

#### Data

* **Name** - **string**
  + If the name **is null or empty,** throw an **ArgumentException** with the message: **"Gym name cannot be null or empty."**
  + All names are unique
* **Capacity** - **int**
  + The **number** of **Athletes** who can exercise in a **Gym**
* **Equipment** - **ICollection<IEquipment>**
* **Athletes** - **ICollection<IAthlete>**
* **EquipmentWeight** - calculated property, which returns **double**
  + How is it calculated: The **sum** of **each equipment’s weight** in the **Gym**

#### Behavior

##### void AddAthlete(IAthlete athlete)

**Adds** an **athlete** in the **gym** if there is **space left** for him/her, otherwise throws an **InvalidOperationException** with the message **"Not enough space in the gym."**.

##### bool RemoveAthlete(IAthlete athlete)

Removes an **athlete** from the **gym**. Returns **true** if the **athlete** is removed successfully, otherwise - **false**.

##### void AddEquipment(IEquipment equipment)

Adds a piece of **equipment** in the **gym**.

##### void Exercise()

The **Exercise()** method **trains** **all athletes**, by calling their **Exercise()** method.

##### string GymInfo()

**Returns** a **string** with **information** aboutthe **gym** in the format below:

**"{gymName} is a {gymType}:****Athletes: {athleteName1}, {athleteName2}, {athleteName3} (…) / No athletes****Equipment total count: {equipmentCount}****Equipment total weight: {equipmentWeight} grams"**

**Note: Do not use** "**\n\r**" **for a new line.**

#### Constructor

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

**string name, int capacity**

#### Child Classes

There are 2 concrete types of **Gym**:

##### BoxingGym

**Up to 15 athletes** can exercise in the **BoxingGym**.

The constructorshould take the following values upon initialization:

**string name**

##### WeightliftingGym

**Up to 20 athletes** can exercise in the **WeightliftingGym**.

The constructorshould take the following values upon initialization:

**string name**

### EquipmentRepository

The **equipment repository** is a **repository** for the **equipment** that is in the **Gym**.

#### Data

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

#### Behavior

##### void Add(IEquipment equipment)

* **Added** **equipment** to the **collection**.

**bool Remove(IEquipment equipment)**

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

**IEquipment FindByType(string equipmentType)**

* **Returns** the **first** **equipment** of the **given type**, if there is one. **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:

* **equipment** - **EquipmentRepository**
* **gyms** - a **collection of IGym**

### Commands

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

#### AddGym Command

##### Parameters

* gymType - string
* gymName - string

##### Functionality

**Adds** a Gym to the gym's collection. **Valid** types of gyms are: " **BoxingGym**" and " **WeightliftingGym**".

* If the **Gym** **type** is **invalid**, **throw an InvalidOperationException** with **the following message:** **"Invalid gym type."**
* If the **Gym** is **added successfully**, **return** the following **message**: **"Successfully added {gymType}."**

#### AddEquipment Command

##### Parameters

* **equipmentType** - **string**

##### Functionality

**Creates** **equipment** of the **given type** and **adds** it to the **EquipmentRepository**. **Valid** types are: "**BoxingGloves**" and "**Kettlebell**".

* If the equipment **type** is **invalid**, throw an **InvalidOperationException** with the message: **"Invalid equipment type."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"Successfully added {equipmentType}."**

#### InsertEquipment Command

##### Parameters

* gymName - string
* equipmentType - string

##### Functionality

**Adds** the desired Equipment to the Gym with the **given name**. You have to remove the Equipment from the EquipmentRepository if the insert is **successful**.

* If there is **no such equipment**, **throw an InvalidOperationException** with **the following message**: **"There isn't equipment of type {equipmentType}."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"Successfully added {equipmentType} to {gymName}."**

#### AddAthlete Command

##### Parameters

* gymName - string
* athleteType - string
* athleteName - string
* motivation - string
* numberOfMedals - int

##### Functionality

**Creates** and **adds** an Athlete to the Gym with the **given name**. **Valid** Athletes types are: "**Boxer**" (an exercise in a "**BoxingGym**"**)**, and "**Weightlifter**" (an exercise in a "**WeightliftingGym**").

**Return** one of the following messages:

* If the Athlete **type** is **invalid**, **throw an InvalidOperationException** with **the following message:** **"Invalid athlete type."**
* If the **Athlete** **cannot exercise** in the given **Gym, return** a string with the following **message: "The gym is not appropriate."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"Successfully added {athleteType} to {gymName}."**

#### TrainAthletes Command

##### Parameters

* **gymName** - **string**

##### Functionality

Exercise all athletes in the Gym with the given name. **Returns** a **string** with information about **how many** athletes did **exercise**, in the following **format**:

* **"Exercise athletes: {athletesCount}."**

#### EquipmentWeight Command

##### Parameters

* **gymName** - **string**

##### Functionality

Calculates the weight of all available equipment of the **Gym** with the given name. It is calculated by the sum of all inserted equipment in the **Gym**.

**Return** a **string** in the following **format**:

* **"The total weight of the equipment in the gym {gymName} is {value} grams."**
  + The **value** should be **formatted** to the **2nd decimal place**!

#### Report Command

##### Functionality

Returns information about each gym. You can use the overridden **GymInfo Gym** method.

**"{gymName} is a {gymType}:  
Athletes: {fullName1}, {fullName2}, {fullName3} (…) / No athletes  
Equipment total count: {equipmentCount}  
Equipment total weight: {equipmentWeight} grams**

**{gymName} is a {gymType}:  
Athletes: {fullName1}, {fullName2}, {fullName3} (…) / No athletes  
Equipment total count: {equipmentCount}  
Equipment total weight: {equipmentWeight} grams**

**(…)"**

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

#### 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:

* **AddGym** **{gymType} {gymName}**
* **AddEquipment** **{equipmentType}**
* **InsertEquipment** **{gymName} {equipmentType}**
* **AddAthlete {gymName} {athleteType} {athleteName} {motivation} {numberOfMedals}**
* **TrainAthletes {gymName}**
* **EquipmentWeight {gymName}**
* **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** |
| **AddGym WeightliftingGym QuadsGym**  **AddGym BoxingGym Gloveworx**  **AddAthlete Gloveworx Boxer Mike-Bodysnatcher-McCallum Positive 10**  **AddAthlete Gloveworx Weightlifter Ray-Merciless-Mercer Intrinsic 8**  **AddGym BoxingGym GothamGym**  **AddAthlete GothamGym Boxer Rubin-Hurricane-Carter Positive 9**  **AddAthlete QuadsGym Wrestler TripleH Leadership 7**  **AddEquipment BoxingGloves**  **InsertEquipment Gloveworx BoxingGloves**  **InsertEquipment QuadsGym Kettlebell**  **AddEquipment Kettlebell**  **InsertEquipment QuadsGym Kettlebell**  **TrainAthletes Gloveworx**  **AddAthlete QuadsGym Weightlifter Intrinsic 5**  **AddAthlete QuadsGym Weightlifter Flex-Wheeler 8**  **AddAthlete QuadsGym Weightlifter Flex-Wheeler Positive -8**  **Report**  **Exit** |
| **Output** |
| **Successfully added WeightliftingGym.**  **Successfully added BoxingGym.**  **Successfully added Boxer to Gloveworx.**  **The gym is not appropriate.**  **Successfully added BoxingGym.**  **Successfully added Boxer to GothamGym.**  **Invalid athlete type.**  **Successfully added BoxingGloves.**  **Successfully added BoxingGloves to Gloveworx.**  **There isn't equipment of type Kettlebell.**  **Successfully added Kettlebell.**  **Successfully added Kettlebell to QuadsGym.**  **Exercise athletes: 1.**  **Athlete name cannot be null or empty.**  **The motivation cannot be null or empty.**  **Athlete's number of medals cannot be below 0.**  **QuadsGym is a WeightliftingGym:**  **Athletes: No athletes**  **Equipment total count: 1**  **Equipment total weight: 10000.00 grams**  **Gloveworx is a BoxingGym:**  **Athletes: Mike-Bodysnatcher-McCallum**  **Equipment total count: 1**  **Equipment total weight: 227.00 grams**  **GothamGym is a BoxingGym:**  **Athletes: Rubin-Hurricane-Carter**  **Equipment total count: 0**  **Equipment total weight: 0.00 grams** |

|  |
| --- |
| **Input** |
| **AddGym WeightliftingGym QuadsGym**  **AddEquipment Kettlebell**  **AddEquipment Kettlebell**  **InsertEquipment QuadsGym Kettlebell**  **InsertEquipment QuadsGym Kettlebell**  **InsertEquipment QuadsGym Kettlebell**  **AddAthlete QuadsGym Weightlifter Geoffrey-Oduor Intrinsic 8**  **AddAthlete QuadsGym Weightlifter Franklin-Atete Leadership 3**  **TrainAthletes QuadsGym**  **AddAthlete QuadsGym Weightlifter Faris-Touairi Extrinsic 3**  **EquipmentWeight QuadsGym**  **TrainAthletes QuadsGym**  **Report**  **Exit** |
| **Output** |
| **Successfully added WeightliftingGym.**  **Successfully added Kettlebell.**  **Successfully added Kettlebell.**  **Successfully added Kettlebell to QuadsGym.**  **Successfully added Kettlebell to QuadsGym.**  **There isn't equipment of type Kettlebell.**  **Successfully added Weightlifter to QuadsGym.**  **Successfully added Weightlifter to QuadsGym.**  **Exercise athletes: 2.**  **Successfully added Weightlifter to QuadsGym.**  **The total weight of the equipment in the gym QuadsGym is 20000.00 grams.**  **Exercise athletes: 3.**  **QuadsGym is a WeightliftingGym:**  **Athletes: Geoffrey-Oduor, Franklin-Atete, Faris-Touairi**  **Equipment total count: 2**  **Equipment total weight: 20000.00 grams** |

## Task 3: Unit Tests (100 points)

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