## C# OOP Exam - 15 December 2019



## Overview

Aquariums are nice and interesting species can live in there. You have to create an **AquaShop** project, which keeps track of the fish in the aquariums. The Aquariums have **Fish** with different environment requirements. Your task is to add, feed and take care of the fish.

## Setup

* Upload **only the** AquaShopproject 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: **Aquarium, Fish, Decoration**. There should also be **DecorationRepository**.

### Decoration

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

#### Data

* **Comfort** - **int**
* **Price** - **decimal**
  + The price of the decoration

#### Constructor

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

int comfort, decimal price

#### Child Classes

There are several concrete types of **Decoration**:

##### Ornament

Has **1 comfort** and its **price** is **5**.

Constructorshould take no values upon initialization.

##### Plant

Has **5 comfort** and its **price** is **10**.

Constructorshould take no values upon initialization.

### Fish

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

#### Data

* **Name** - **string**
  + If the name **is null or whitespace,** throw an **ArgumentException** with message: "Fish name cannot be null or empty."
  + All names are unique
* **Species** - **string**
  + If the species **is null or whitespace,** throw an **ArgumentException** with message: "Fish species cannot be null or empty."
* **Size** - **int**
  + The size of the **Fish**
* **Price** - **decimal**
  + The price of the **Fish**
  + If the price is below or equal **0,** throw an **ArgumentException** with message:

"Fish price cannot be below or equal to 0."

#### Behavior

##### abstract void Eat()

The **Eat()** method increases the **Fish**’s size.

#### Constructor

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

string name, string species, decimal price

#### Child Classes

There are several concrete types of **Fish**:

##### FreshwaterFish

Has **3 initial size**.

**Can only live in FreshwaterAquarium!**

Constructorshould take the following values upon initialization:

string name, string species, decimal price

###### **Behavior**

**void Eat()**

* The method **increases** the fish’s size by **3**.

##### SaltwaterFish

Has **5 initial size**.

**Can only live in SaltwaterAquarium!**

Constructorshould take the following values upon initialization:

string name, string species, decimal price

###### **Behavior**

**void Eat()**

* The method **increases** the fish’s size by **2**.

### Aquarium

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

#### Data

* **Name** - **string**
  + If the name **is null or whitespace,** throw an **ArgumentException** with message: "Aquarium name cannot be null or empty."
  + All names are unique
* **Capacity** - **int**
  + The **number** of **Fish** аn **Aquarium** **can have**
* **Decorations** - **ICollection<IDecoration>**
* **Fish** - **ICollection<IFish>**
* **Comfort** - calculated property, which returns **int**
  + How is it calculated: The **sum** of **each decoration’s comfort** in the **Aquarium**

#### Behavior

##### void AddFish(IFish fish)

**Adds** a **Fish** in the **Aquarium** if there is **capacity** for it ~~and if the~~ **~~water is suitable~~** ~~for the~~ **~~Fish~~**, otherwise throw an **InvalidOperationException** with message "Not enough capacity.";

##### bool RemoveFish(IFish fish)

Removes a **Fish** from the **Aquarium**. Returns **true** if the **Fish** is removed successfully, otherwise - **false**.

##### void AddDecoration(IDecoration decoration)

Adds a **Decoration** in the **Aquarium**.

##### void Feed()

The **Feed()** method **feeds** **all fish**, calls their **Eat()** method.

##### string GetInfo()

**Returns** a **string** with **information** about the **Aquarium** in the format below. If the **Aquarium doesn't have fish**, print **"none"** instead.

"{aquariumName} ({aquariumType}):  
Fish: {fishName1}, {fishName2}, {fishName3} (…) / none  
Decorations: {decorationsCount}  
Comfort: {aquariumComfort}"

#### Constructor

An **Aquarium** should take the following values upon initialization:

string name, int capacity

#### Child Classes

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

##### FreshwaterAquarium

Has **50 capacity**.

Constructorshould take the following values upon initialization:

string name

##### SaltwaterAquarium

Has **25 capacity**

Constructorshould take the following values upon initialization:

string name

### DecorationRepository

The **decoration repository** is a **repository** for the **decorations** that are in the **AquaShop**.

#### Data

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

#### Behavior

##### void Add(IDecoration decoration)

* **Adds** a **decoration** in the **collection**.

**bool Remove(IDecoration decoration)**

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

**IDecoration FindByType(string type)**

* **Returns** the **first** **decoration** of the **given type**, if there is. **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 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.

### Data

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

* **decorations** - **DecorationRepository**
* **aquariums** - **collection of IAquarium**

### Commands

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

#### AddAquarium Command

##### Parameters

* aquariumType - string
* aquariumName - string

##### Functionality

**Adds** an Aquarium. **Valid** types are: "**FreshwaterAquarium**" and "**SaltwaterAquarium**".

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

* "Invalid aquarium type."

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

* "Successfully added {aquariumType}."

#### AddDecoration Command

##### Parameters

* **type** - **string**

##### Functionality

**Creates** a **decoration** of the **given type** and **adds** it to the **DecorationRepository**. **Valid** types are: "**Ornament**" and "**Plant**". If the decoration **type** is **invalid**, throw an **InvalidOperationException** with message:

* "Invalid decoration type."

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

* "Successfully added {decorationType}."

#### InsertDecoration Command

##### Parameters

* aquariumName - string
* decorationType - string

##### Functionality

**Adds** the desired Decoration to the Aquarium with the **given name**. You have to remove the Decoration from the DecorationRepository if the insert is **successful**.

If there is **no such decoration**, you have to **throw an InvalidOperationException** with **the following message**:

* "There isn't a decoration of type {decorationType}."

If **no errors** are **thrown**, **return** a string with the following message "Successfully added {decorationType} to {aquariumName}.".

#### AddFish Command

##### Parameters

* aquariumName - string
* fishType - string
* fishName - string
* fishSpecies - string
* price - decimal

##### Functionality

and if the **water is suitable** for the **Fish**

**Adds** the desired Fish to the Aquarium with the **given name**. **Valid** Fish types are: "**FreshwaterFish**", "**SaltwaterFish**".

If the **Fish** **type** is **invalid**, you have to **throw an InvalidOperationException** with **the following message** "Invalid fish type.".

If **no errors** are **thrown**, **return** one of the following messages:

* "Water not suitable." - if the **Fish** **cannot live** in the **Aquarium**
* "Successfully added {fishType} to {aquariumName}." - if the **Fish** is **added successfully** in the **Aquarium**

#### FeedFish Command

##### Parameters

* **aquariumName** - **string**

##### Functionality

Feeds all **Fish** in the **Aquarium** with the given name.

**Returns** a **string** with information about **how many fish** were **fed**, in the following **format**:

* "Fish fed: {fedCount}"

#### CalculateValue Command

##### Parameters

* **aquariumName** - **string**

##### Functionality

Calculates the value of the **Aquarium** with the given name. It is calculated by the sum of all **Fish**’s and **Decorations**’ prices in the **Aquarium**.

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

* "The value of Aquarium {aquariumName} is {value}."
  + The **value** should be **formatted** to the **2nd decimal place**!

#### Report Command

##### Functionality

Returns information about each aquarium. You can use the overridden **GetInfo Aquarium** method.

"{aquariumName} ({aquariumType}):  
Fish: {fishName1}, {fishName2}, {fishName3} (…) / none  
Decorations: {decorationsCount}  
Comfort: {aquariumComfort}

{aquariumName} ({aquariumType}):  
Fish: {fishName1}, {fishName2}, {fishName3} (…) / none  
Decorations: {decorationsCount}  
Comfort: {aquariumComfort}

(…)"

**Note: Use \r\n or Environment.NewLine for a new line.**

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

#### Input

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

* **AddAquarium** **{aquariumType} {aquariumName}**
* **AddDecoration** **{decorationType}**
* **InsertDecoration** **{aquariumName} {decorationType}**
* **AddFish {aquariumName} {fishType} {fishName} {fishSpecies} {price}**
* **FeedFish {aquariumName}**
* **CalculateValue {aquariumName}**
* **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** |
| **AddAquarium SaltwaterAquarium Underworld**  **AddAquarium FreshwaterAquarium Swamp**  **AddFish Underworld FreshwaterFish Nemo Clownfish 13.40**  **AddFish Underworld SaltwaterFish Nemo Clownfish 13.40**  **AddAquarium FreshwaterAquarium Riverworld**  **AddFish Riverworld FreshwaterFish Emerald Catfish 7.32**  **AddFish Underworld SweetwaterFish Diamond Catfish 3.50**  **AddDecoration Plant**  **InsertDecoration Riverworld Plant**  **InsertDecoration Underworld Plant**  **AddDecoration Plant**  **InsertDecoration Underworld Plant**  **FeedFish Riverworld**  **AddFish Riverworld FreshwaterFish Species 20**  **AddFish Riverworld FreshwaterFish Name 20**  **AddFish Riverworld FreshwaterFish Name Species -10**  **Report**  **Exit** |
| **Output** |
| **Successfully added SaltwaterAquarium.**  **Successfully added FreshwaterAquarium.**  **Water not suitable.**  **Successfully added SaltwaterFish to Underworld.**  **Successfully added FreshwaterAquarium.**  **Successfully added FreshwaterFish to Riverworld.**  **Invalid fish type.**  **Successfully added Plant.**  **Successfully added Plant to Riverworld.**  **There isn't a decoration of type Plant.**  **Successfully added Plant.**  **Successfully added Plant to Underworld.**  **Fish fed: 1**  **Fish name cannot be null or empty.**  **Fish species cannot be null or empty.**  **Fish price cannot be below or equal to 0.**  **Underworld (SaltwaterAquarium):**  **Fish: Nemo**  **Decorations: 1**  **Comfort: 5**  **Swamp (FreshwaterAquarium):**  **Fish: none**  **Decorations: 0**  **Comfort: 0**  **Riverworld (FreshwaterAquarium):**  **Fish: Emerald**  **Decorations: 1**  **Comfort: 5** |

|  |
| --- |
| **Input** |
| **AddAquarium SaltwaterAquarium DangerZone**  **AddDecoration Plant**  **AddDecoration Plant**  **AddDecoration Ornament**  **InsertDecoration DangerZone Plant**  **InsertDecoration DangerZone Plant**  **InsertDecoration DangerZone Ornament**  **AddFish DangerZone SaltwaterFish Curibou Angelfish 22.33**  **AddFish DangerZone SaltwaterFish Devil Anglerfish 48.84**  **FeedFish DangerZone**  **AddFish DangerZone EuryhalineFish Greeny Chromide 9.99**  **CalculateValue DangerZone**  **FeedFish DangerZone**  **Report**  **Exit** |
| **Output** |
| **Successfully added SaltwaterAquarium.**  **Successfully added Plant.**  **Successfully added Plant.**  **Successfully added Ornament.**  **Successfully added Plant to DangerZone.**  **Successfully added Plant to DangerZone.**  **Successfully added Ornament to DangerZone.**  **Successfully added SaltwaterFish to DangerZone.**  **Successfully added SaltwaterFish to DangerZone.**  **Fish fed: 2**  **Invalid fish type.**  **The value of Aquarium DangerZone is 96.17.**  **Fish fed: 2**  **DangerZone (SaltwaterAquarium):**  **Fish: Curibou, Devil**  **Decorations: 3**  **Comfort: 11** |

## Task 3: Unit Tests (100 points)

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