## 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 AquaShop** project 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
* Качете само проекта AquaShop при всеки проблем, с изключение на тестовете на единица
* ⦁ Не променяйте интерфейсите или техните пространства от имена
* ⦁ Използвайте силно сцепление и разхлабена връзка
* ⦁ Използвайте наследяване и предоставените интерфейси, когато е възможно.
* ⦁ Това включва конструктори, параметри на метода и видове връщане
* ⦁ Не нарушавайте реализациите на вашия интерфейс, като добавите повече публични методи или свойства в конкретния клас, отколкото интерфейсът е определил
* ⦁ Уверете се, че нямате публични полета никъде

# 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**.

За тази задача логиката за оценка в методите не е включена.

Дават ви интерфейси и трябва да внедрите тяхната функционалност в правилните класове.

В приложението има 3 вида образувания: Аквариум, Риби, Декорация. Трябва да има и DecorationRepository

# Decoration

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

Декорацията е основен клас на всеки тип декорация и тя не трябва да може да бъде интензивна.

**Data**

* **Comfort** - **int**
* **Price** - **decimal**
* The price of the decoration
* Цената на декорацията

**Constructor**

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

int comfort, decimal price

# Child Classes

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

Има няколко конкретни вида декорация:

# Ornament

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

Има 1 комфорт, а цената му е 5.

Constructorshould take no values upon initialization.

Конструкторът не трябва да приема стойности при инициализация.

# 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**.

Рибата е основен клас на всеки вид риба и тя не трябва да може да бъде инстанцирана.

**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
* Ако името е нулево или бяло пространство, хвърлете ArgumentException със съобщение: "Името на рибата не може да бъде нулево или празно."
* ⦁ Всички имена са уникални
* **Species** - **string**
* If the species **is null or whitespace,** throw an **ArgumentException** with message: "Fish species cannot be null or empty."
* Ако видът е нулев или бяло пространство, хвърлете ArgumentException със съобщение: "Видовете риби не могат да бъдат нули или празни."
* **Size** - **int**
* The size of the **Fish**
* **Размерът на рибата**
* **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."

Цената на Рибите

⦁ Ако цената е под или равна на 0, хвърлете ArgumentException със съобщение:

„Цената на рибата не може да бъде под или равна на 0.“

**Behavior**

# abstract void Eat()

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

Методът Eat () увеличава размера на рибата.

**Constructor**

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

Рибата трябва да приеме следните стойности при инициализация:

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!**

**Има 3 първоначални размера.**

**Може да живее само в 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**.
* Методът увеличава размера на рибата с 3.

# SaltwaterFish

Has **5 initial size**.

**Can only live in SaltwaterAquarium!**

**Има 5 първоначални размера.**

**Може да живее само в Солената вода Аквариум!**

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**.
* Методът увеличава размера на рибата с 2.

# Aquarium

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

Аквариумът е основен клас на всеки тип Аквариум и не трябва да може да бъде инстанциран.

**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
* Ако името е нулево или бяло пространство, хвърлете ArgumentException със съобщение: "Името на аквариума не може да бъде нищожно или празно."
* ⦁ Всички имена са уникални
* **Capacity** - **int**
* The **number** of **Fish** аn **Aquarium** **can have**
* **Броят на рибите и аквариума може да има**
* **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**
* Комфорт - изчислено свойство, което връща вх
* ⦁ Как се изчислява: сумата от комфорта на всяка декорация в Аквариума

**Behavior**

# void AddFish(IFish fish)

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

Добавя риба в аквариума, ако има капацитет за нея, в противен случай хвърлете InvalidOperationException със съобщение „Не е достатъчен капацитет.“;

# bool RemoveFish(IFish fish)

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

Премахва риба от аквариума. Връща истина, ако Рибата е премахната успешно, в противен случай - невярна.

# void AddDecoration(IDecoration decoration)

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

Добавя украса в аквариума.

**void Feed()**

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

Методът Feed () захранва всички риби, извиква техния метод Eat ().

# string GetInfo()

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

Връща низ с информация за Аквариума във формата по-долу. Ако Аквариумът няма риба, отпечатайте вместо „няма“.

"{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**.

Хранилището за декорации е хранилище за декорациите, които са в AquaShop

## Data

* **Models** - **a** **collection of decorations (unmodifiable)**
* **Модели - колекция от декорации (немодифицируема)**

## 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**.
* Премахва декорация от колекцията. Връща true, ако изтриването е било успешно, в противен случай - false.

## IDecoration FindByType(string type)

* **Returns** the **first** **decoration** of the **given type**, if there is. **Otherwise**, returns **null**.
* Връща първата декорация от дадения тип, ако има такава. В противен случай връща нула.

# 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 **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section.

изнес логиката на програмата трябва да се концентрира около няколко команди. Дават ви интерфейси, които трябва да внедрите в правилните класове.

Забележка: Класът на контролера НЕ трябва да обработва изключения! Тестовете са създадени да очакват изключения, а не съобщения!

Първият интерфейс е IController. Трябва да създадете клас Controller, който реализира интерфейса и реализира всичките му методи. Конструкторът на Controller не взема никакви аргументи. Дадените методи трябва да имат логиката, описана за всеки в секцията Команди.

### Data

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

Трябва да следите някои неща, ето защо се нуждаете от някои частни полета във вашия клас контролер:

* **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**!

Има няколко команди, които контролират бизнес логиката на приложението. Те са посочени по-долу. Името на Аквариума, предадено на методите, винаги ще бъде валидно!

### AddAquarium Command

**Parameters**

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

**Functionality**

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

Добавя аквариум. Валидни типове са: „FreshwaterAquarium“ и „SaltwaterAquarium“.

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

Ако типът Аквариум е невалиден, трябва да изхвърлите InvalidOperationException със следното съобщение:

* "Invalid aquarium type."

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

Ако аквариумът е добавен успешно, методът трябва да върне следния низ:

* "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:

Създава декорация от дадения тип и я добавя към DecorationRepository. Валидни типове са: „Орнамент“ и „Растение“. Ако типът декорация е невалиден, хвърлете InvalidOperationException със съобщение:

* "Invalid decoration type."

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

Методът трябва да върне следния низ, ако операцията е успешна:

* "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**.

Добавя желаната украса към Аквариума с даденото име. Трябва да премахнете декорацията от DecorationRepository, ако вмъкването е успешно.

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

Ако няма такава украса, трябва да хвърлите InvalidOperationException със следното съобщение:

* "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}.".

Ако не бъдат изхвърлени грешки, върнете низ със следното съобщение „Успешно добавено {decorationType} към {aquariumName}.“.

### AddFish Command

**Parameters**

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

**Functionality**

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

Добавя желаната Риба в Аквариума с даденото име. Валидни видове риба са: „Сладководна риба“, „Солена вода“.

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

Ако типът Риба е невалиден, трябва да хвърлите InvalidOperationException със следното съобщение "Невалиден тип риба."

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

Ако няма изхвърлени грешки, върнете едно от следните съобщения:

* "Water not suitable." - if the **Fish** **cannot live** in the **Aquarium**
* **ако Рибите не могат да живеят в Аквариума**
* "Successfully added {fishType} to {aquariumName}." - if the **Fish** is **added successfully** in the **Aquarium**
* ако Рибите се добавят успешно в Аквариума
* ⦁

### FeedFish Command

**Parameters**

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

**Functionality**

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

Храни всички риби в аквариума с даденото име.

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

Връща низ с информация за това колко риби са били хранени, в следния формат:

* "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**.

Изчислява стойността на Аквариума с даденото име. Изчислява се от сбора от всички цени на Рибите и декорациите в Аквариума.

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

Върнете низ в следния формат:

* "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.

Връща информация за всеки аквариум. Можете да използвате метода на GetInfo Aquarium.

"{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.

Осигурен ви е един интерфейс, който ще ви помогне с правилния процес на изпълнение на вашата програма. Интерфейсът е IEngine и класът, реализиращ този интерфейс, трябва да прочете входа и когато програмата завърши, този клас трябва да отпечата резултата.

Предоставен ви е класът Engine с записана логика в него. За да бъде съставен кодът, някои части се коментират, не забравяйте да ги коментирате. Блокът за пробвания също се коментира, за да може програмата да хвърля изключения и да ги видите, коментирайте, когато сте готови и с това.

**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.

Отпечатайте изхода от всяка команда, когато бъде издадена. Ако по време на изпълнение на някоя от командите е хвърлено изключение, отпечатайте съобщението за изключение.

**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!