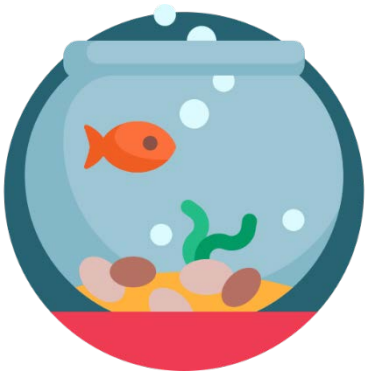
Java OOP Exam – 15 Dec 2019



1. 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.

2. Setup

- Upload **only the AquaShop** package in every task **except Unit Tests**
- **Do not modify the interfaces or their packages**
- 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** in the concrete class than the interface has defined
- Make sure you have **no public fields** anywhere

3. Task 1: Structure (50 points)

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`**.

BaseDecoration

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

Data

- **`comfort`** - **`int`**
- **`price`** - **`double`**
    - The price of the decoration

Constructor

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

(int comfort, double price)

Child Classes

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

Ornament

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

Constructor should take no values upon initialization.

Plant

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

Constructor should take no values upon initialization.

BaseFish

BaseFish 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 **NullPointerException** with message: **"Fish name cannot be null or empty."**
  - All names are unique
- **species** - **String**
  - If the species **is null or whitespace,** throw an **NullPointerException** with message: **"Fish species cannot be null or empty."**
- **size** - **int**
  - The size of the **Fish**
- **price** - **double**
  - The price of the **Fish**
  - If the price is below or equal **0,** throw an **IllegalArgumentException** with message: **"Fish price cannot be below or equal to 0."**

Behavior

void eat()

The **eat()** method increases the **Fish**'s size. Keep in mind that some types of **Fish** can implement the method in a different way.

- The method **increases** the fish's size by **5**.

Constructor

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

(String name, String species, double price)

Child Classes

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

FreshwaterFish

Has **3 initial size**.

Can only live in `FreshwaterAquarium`!

Constructor should take the following values upon initialization:

```
(String name, String species, double price)
```

Behavior

`void eat()`

- The method **increases** the fish's size by **3**.

SaltwaterFish

Has **5 initial size**.

Can only live in `SaltwaterAquarium`!

Constructor should take the following values upon initialization:

```
(String name, String species, double price)
```

Behavior

`void eat()`

- The method **increases** the fish's size by **2**.

BaseAquarium

`BaseAquarium` 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 **`NullPointerException`** with message: **`"Aquarium name cannot be null or empty."`**
  - All names are unique
- **`capacity`** - **`int`**
  - The **number** of **`Fish`** an **`Aquarium`** can have
- **`decorations`** - **`Collection<Decoration>`**
- **`fish`** - **`Collection<Fish>`**

Behavior

Constructor

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

```
(String name, int capacity)
```

`int calculateComfort()`

Returns the sum of **each decoration's comfort** in the **`Aquarium`**

`void addFish(Fish fish)`

Adds a **`Fish`** in the **`Aquarium`** if there is **capacity** for it

if there is **not enough capacity** to **add** the **`Fish`** in the **`Aquarium`** **throw an `IllegalStateException`** with **the following message:**

- **`"Not enough capacity."`**

`void removeFish(Fish fish)`

Removes a **`Fish`** from the **`Aquarium`**.

`void addDecoration(Decoration decoration)`

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

`void feed()`

The **`feed()`** method **feeds all fishes** in the aquarium.

`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}"
```

Child Classes

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

FreshwaterAquarium

Has **50 capacity**

Constructor should take the following values upon initialization:

`String name`

SaltwaterAquarium

Has **25 capacity**

Constructor should take the following values upon initialization:

`String name`

DecorationRepository

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

Data

- **`decorations`** - **`Collection<Decoration>`** **(unmodifiable)**

Behavior

`void add(Decoration decoration)`

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

`boolean remove(Decoration decoration)`

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

`Decoration 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 `ControllerImpl` class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!

The first interface is **`Controller`**. You must create a **`ControllerImpl`** class, which implements the interface and implements all its methods. The constructor of **`ControllerImpl`** does **not take** any **arguments**. The given methods should have the following logic:

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 Aquarium`**

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 `IllegalArgumentException`** 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 **IllegalArgumentException** 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 IllegalArgumentException** with **the following message**:

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

If **no exceptions** are **thrown return** the **String**:

- **"Successfully added {decorationType} to {aquariumName}."**

AddFish Command

Parameters

- **aquariumName** - **String**
- **fishType** - **String**
- **fishName** - **String**
- **fishSpecies** - **String**
- **price** - **double**

Functionality

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 IllegalArgumentException** with **the following message**:

- **"Invalid fish type."** - if the **Fish type** is **invalid**

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

- **"Not enough capacity."** - if there is **not enough capacity** to **add** the **Fish** in the **Aquarium**

- **"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 **successfully 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 \n or System.lineSeparator() 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 **`Engine`** and the class implementing this interface should read the input and when the program finishes, this class should print the output.

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 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`
`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`
`The value of Aquarium DangerZone is 96.17.`
`Fish fed: 2`
`DangerZone (SaltwaterAquarium):`
`Fish: Curibou Devil`

```
Decorations: 3
Comfort: 11
```

Input
AddAquarium SaltwaterAquarium Underworld 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 AddFish Underworld EuryhalineFish Chico Stingray 33.99 AddFish Riverworld EuryhalineFish Bully Shark 48.99 AddDecoration Plant InsertDecoration Riverworld Plant InsertDecoration Underworld Plant AddDecoration Plant InsertDecoration Underworld Plant FeedFish Riverworld FeedFish Riverworld AddFish Riverworld FreshwaterFish Name Species -10 Report Exit
Output
Successfully added SaltwaterAquarium. Water not suitable. Successfully added SaltwaterFish to Underworld. Successfully added FreshwaterAquarium. Successfully added FreshwaterFish to Riverworld. Invalid fish type. Invalid fish type. 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 fed: 1 Fish price cannot be below or equal to 0. Underworld (SaltwaterAquarium): Fish: Nemo Decorations: 1 Comfort: 5 Riverworld (FreshwaterAquarium): Fish: Emerald Decorations: 1 Comfort: 5