# Java OOP Retake Exam – 16 April 2024

1. **Overview**

You have to create a **dolphinarium** project, which keeps track of the dolphins in the basins in the dolphinarium. The **Pools** have **dolphins** with different environmental needs. Your task is to add, feed, and take care of the dolphins.

## Setup

* Upload **only the dolphinarium** 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.

## Task 1: Structure (50 points)

You are given **3** interfaces and you must implement their functionalities in the **correct classes**.

There are **3** types of entities in the application: **Pool, Dolphin, and Food**. There should also be **FoodRepository**.

### Food

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

#### Data

* **calories** - **int**
  + The amount of calories in each food.

#### Constructor

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

(int calories)

#### Child Classes

There are three concrete types of **Food**:

##### Squid

Has **175 calories**.

The constructorshould take no values upon initialization.

##### Herring

Has **200 calories**.

The constructorshould take no values upon initialization.

##### Mackerel

Has **305 calories**.

The constructorshould take no values upon initialization.

### Dolphin

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

#### Data

* **name** - **String**
  + If the name **is null or whitespace,** throw a **NullPointerException** with a message:

"**Dolphin name cannot be null or empty.**"

* + All names are unique.
* **energy**- **int**
  + The energy of the **Dolphin.**
  + It is guaranteed that you will receive only positive values
  + If the energy drops below **0** (zero) **set** it to 0 (zero)

#### Behavior

##### void jump()

The **jump()** method **decreases** the **Dolphin’s** energy initially by **10** units. Each **Dolphin** implements the method differently and **reduces** the energy by **additional** units depending on its type.

##### void eat(Food food)

The **eat()** method increases the **Dolphin’s** energy.

* The method **increases** the dolphin’s energy by the **value** of calories contained in the food he ate:
* If a dolphin eats **Squid** – his energy increases by **175**;
* If a dolphin eats a **Herring** – his energy increases by **200**;
* If a dolphin eats **Mackerel** – his energy increases by **305**;

#### Constructor

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

(String name, int energy)

#### Child Classes

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

##### BottleNoseDolphin

**Can only swim in DeepWaterPool!**

The constructorshould take the following values upon initialization:

**(String name, int energy)**

#### Behavior

**void jump()**

* The method **decreases** the dolphin’s energy by an additional **190 units**

##### SpottedDolphin

**Can swim in DeepWaterPool as well as in ShallowWaterPool.**

The constructorshould take the following values upon initialization:

**(String name, int energy)**

#### Behavior

**void jump()**

* The method **decreases** the dolphin’s energy by an additional **90 units**.

##### SpinnerDolphin

**Can only swim in ShallowWaterPool!**

The constructorshould take the following values upon initialization:

**(String name, int energy)**

#### Behavior

**void jump()**

* The method **decreases** the dolphin’s energy by an additional **40 units**.

### Pool

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

#### Data

* **name** - **String**
  + If the name **is null or whitespace,** throw a **NullPointerException** with a message:

"**Pool name cannot be null or empty.**"

* + All names are unique.
* **capacity** - **int**
  + The **number** of **Dolphin** аn **Pool** **can have.**
* **foods** - **Collection<Food>**
* **dolphins** - **Collection<Dolphin>**

#### Behavior

##### Constructor

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

**(String name, int capacity)**

##### void addDolphin(Dolphin dolphin)

**Adds** an **Dolphin** in the **Pool** if there is the **capacity** for it.

If there is **not enough capacity** to **add** the **Dolphin** to the **Pool** **throw an IllegalStateException** with **the following message:**

* **"Not enough capacity."**

If the **energy** of the dolphin is below or equal to **0,** throw an **IllegalArgumentException** with the message:

"**Dolphin energy cannot be below or equal to 0.**"

##### void removeDolphin(Dolphin dolphin)

Removes an **Dolphin** from the **Pool**.

##### void addFood(Food food)

Adds a **Food** in the **Pool**.

#### Child Classes

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

##### DeepWaterPool

Has **5 capacity.**

The constructorshould take the following values upon initialization:

**(String name)**

##### ShallowWaterPool

Has **2 capacity.**

The constructorshould take the following values upon initialization:

**(String name)**

### FoodRepository

The **FoodRepositoryImpl** is a **repository** for the **foods** that are in the **Pool**.

#### Data

* foods - **Collection<Food>**

#### Behavior

**void add(Food food)**

* **Adds** **food** to the **collection**. It is possible to add the same type of food more than once.

**boolean remove(Food food)**

* **Removes** **food** from the **collection**. **Returns true** if the deletion was **successful**, **otherwise** - **false**.

**Food findByType(String type)**

* **Returns** the **first** **food** of the **given type**, if there is one. **Otherwise**, returns **null**.

**Hint:** You can use - **getClass().getSimpleName()** method to determine the food.

## 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 ControllerImplclass, 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:

* **foodRepository** - **FoodRepository**
* **pools** - a **Collection of Pool**

### Commands

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

#### AddPool Command

##### Parameters

* poolType - String
* poolName - String

##### Functionality

**Adds** an Poolin the collection. **Valid** types are: "**DeepWaterPool**" and "**ShallowWaterPool**".

If the **Pool** **type** is **invalid**, you have to **throw a NullPointerException** with **the following message:**

* **"Invalid Pool type."**

If there is a **Pool** with the **same** name, you have to **throw a NullPointerException** with **the following message:**

* **"Pool already exists."**

If the **Pool** is **added successfully**, the method should **return** the following **String**:

* **"Successfully added {poolType} {poolName}."**

#### BuyFood Command

##### Parameters

* **foodType** - **String**

##### Functionality

**Buys** a **food** of the **given type** and **adds** it to the **FoodRepository**. **Valid** types are: **"Squid", "Herring"** and "**Mackerel**". If the food **type** is **invalid**, throw an **IllegalArgumentException** with the message:

* **"Invalid food type."**

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

* **"Successfully bought food {foodType}."**

It's acceptable to buy food from the same type more than once.

#### AddFoodToPool Command

##### Parameters

* poolName - String
* foodType - String

##### Functionality

**Adds** the desired Food to the Pool with the **given name**. You must remove the **Food** from the **FoodRepository** if the insert **succeeds**.

If there is **no such food**, in the **FoodRepository** you have to **throw an IllegalArgumentException** with **the following message**:

* **"There isn't a food of type {**foodType} in repository**."**

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

* **"Successfully added {foodType} to {poolName}."**

#### AddDolphin Command

##### Parameters

* poolName - String
* dolphinType - String
* dolphinName - String
* energy - int

##### Functionality

**Adds** the desired Dolphin to the Pool with the **given name**. **Valid** Dolphin types are "**BottleNoseDolphin**", "**SpottedDolphin**" and **"SpinnerDolphin"**.

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

* **"Invalid dolphin type."**

If the **Dolphin** **name** already **exists**,you have to **throw an IllegalArgumentException** with **the following message**:

* **"Dolphin already exists."**

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

* **"This pool is not suitable."** - if the **Dolphin** **cannot swim** in the **Pool**
* **"Successfully added {dolphinType} {dolphinName} to {poolName}."** - if the **Dolphin** is **added successfully** to the **Pool**

#### FeedDolphins Command

##### Parameters

* **poolName** – **String**
* **foodType - String**

##### Functionality

Feeds all **Dolphin** in the **Pool** with the given name.

If there is such **foodType** **added** in the given pool, **returns** a **string** with information about **how many dolphins** were **successfully fed**, in the following **format**:

* **"{fedDolphinsCount} dolphin/s in pool {poolName} was/were fed."**

**otherwise** **throw an IllegalArgumentException** with **the following message**:

* **"There is no such food for this pool."**

After a successful feeding, you have to **remove** the food that dolphins ate from **foods** collection of the given pool.

**Hint:** You can use - **getClass().getSimpleName()** method to determine the food.

#### PlayWithDolphins Command

##### Parameters

* **poolName** – **String**

##### Functionality

This command makes the dolphins start playing by **jumping**. Each **jump** reduced the dolphin's energy (*each dolphin in the given pool jumps once with every command*) by an amount **depending** on the type of dolphin.

If the dolphin's **energy** reaches **0** (zero) or less you have to **remove** the current dolphin from the pool. Keep track of removed dolphins.

Finally, print the message:

* **"There was a play with dolphin/s in {poolName}. {countOfRemovedDolphins} dolphins was/were removed!"**

If there are no dolphins in the given pool **throw an IllegalArgumentException** with **the following message:**

* **"There are no dolphins to play with."**

**Hint:** You can use **Iterator**.

#### GetStatistics Command

##### Functionality

Returns information about each **Pool**.

**"Dolphins in pool {poolName}:**

**{dolphinName1 - dolphinEnergy1}, {dolphinName2 – dolphinEnergy2}, {dolphinName3 – dolphinEnergy3}, (…)"**If there are **no** dolphins in some pool, print:

**"Dolphins in pool {poolName}:**

**none"**

#### 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:

* **AddPool** **{poolType} {poolName}**
* **BuyFood** **{foodType}**
* **AddFoodToPool {poolName} {foodType}**
* **AddDolphin {poolName} {dolphinType} {dolphinName} {energy}**
* **FeedDolphins {String poolName}, {String food}**
* **PlayWithDolphins {String poolName}**
* **GetStatistics**
* **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** |
| **AddPool DeepWaterPool Cancun**  **AddPool ShallowWaterPool MarineMagic**  **AddPool ShallowWaterPool Corals**  **AddDolphin Cancun BottleNoseDolphin Oum 800**  **AddDolphin MarineMagic SpinnerDolphin Orca 100**  **AddDolphin Cancun BottleNoseDolphin Spike 400**  **AddDolphin Cancun BottleNoseDolphin Henry 0**  **AddDolphin Corals SpinnerDolphin Mike 50**  **BuyFood Mackerel**  **BuyFood Herring**  **BuyFood Herring**  **AddFoodToPool Cancun Herring**  **AddFoodToPool Cancun Mackerel**  **FeedDolphins Cancun Herring**  **PlayWithDolphins Cancun**  **PlayWithDolphins Cancun**  **AddFoodToPool MarineMagic Herring**  **PlayWithDolphins MarineMagic**  **FeedDolphins MarineMagic Mackerel**  **PlayWithDolphins MarineMagic**  **BuyFood Squid**  **AddFoodToPool Corals Squid**  **FeedDolphins Corals Mackerel**  **FeedDolphins Corals Squid**  **FeedDolphins Corals Squid**  **PlayWithDolphins Corals**  **GetStatistics**  **Exit** |
| **Output** |
| **Successfully added DeepWaterPool Cancun.**  **Successfully added ShallowWaterPool MarineMagic.**  **Successfully added ShallowWaterPool Corals.**  **Successfully added BottleNoseDolphin Oum to Cancun.**  **Successfully added SpinnerDolphin Orca to MarineMagic.**  **Successfully added BottleNoseDolphin Spike to Cancun.**  **Dolphin energy cannot be below or equal to 0.**  **Successfully added SpinnerDolphin Mike to Corals.**  **Successfully bought food Mackerel.**  **Successfully bought food Herring.**  **Successfully bought food Herring.**  **Successfully added food Herring to Cancun.**  **Successfully added food Mackerel to Cancun.**  **2 dolphin/s in pool Cancun was/were fed.**  **There was a play with dolphin/s in Cancun. 0 dolphins was/were removed!**  **There was a play with dolphin/s in Cancun. 0 dolphins was/were removed!**  **Successfully added food Herring to MarineMagic.**  **There was a play with dolphin/s in MarineMagic. 0 dolphins was/were removed!**  **There is no such food for this pool.**  **There was a play with dolphin/s in MarineMagic. 1 dolphins was/were removed!**  **Successfully bought food Squid.**  **Successfully added food Squid to Corals.**  **There is no such food for this pool.**  **1 dolphin/s in pool Corals was/were fed.**  **There is no such food for this pool.**  **There was a play with dolphin/s in Corals. 0 dolphins was/were removed!**  **Dolphins in pool Cancun:**  **Oum - 600, Spike - 200**  **Dolphins in pool MarineMagic:**  **none**  **Dolphins in pool Corals:**  **Mike - 175** |
| **Input** |
| **AddPool DeepWaterPool SeaDream**  **AddPool DeepWaterPool SeaDream**  **AddPool ShallowWaterPool OceanCurrent**  **AddPool DeepWaterPool BlueAbyss**  **AddDolphin SeaDream BottleNoseDolphin Rick 600**  **AddDolphin OceanCurrent SpinnerDolphin Spinny 400**  **AddDolphin SeaDream BottleNoseDolphin Rick 60**  **FeedDolphins SeaDream Mackerel**  **AddFoodToPool OceanCurrent Herring**  **BuyFood Mackerel**  **BuyFood Herring**  **BuyFood Herring**  **FeedDolphins SeaDream Mackerel**  **AddFoodToPool SeaDream Herring**  **FeedDolphins SeaDream Herring**  **PlayWithDolphins SeaDream**  **PlayWithDolphins BlueAbyss**  **GetStatistics**  **Exit** |
| **Output** |
| **Successfully added DeepWaterPool SeaDream.**  **Pool already exists.**  **Successfully added ShallowWaterPool OceanCurrent.**  **Successfully added DeepWaterPool BlueAbyss.**  **Successfully added BottleNoseDolphin Rick to SeaDream.**  **Successfully added SpinnerDolphin Spinny to OceanCurrent.**  **Dolphin already exists.**  **There is no such food for this pool.**  **There isn't a food of type Herring in repository.**  **Successfully bought food Mackerel.**  **Successfully bought food Herring.**  **Successfully bought food Herring.**  **There is no such food for this pool.**  **Successfully added food Herring to SeaDream.**  **1 dolphin/s in pool SeaDream was/were fed.**  **There was a play with dolphin/s in SeaDream. 0 dolphins was/were removed!**  **There are no dolphins to play with.**  **Dolphins in pool SeaDream:**  **Rick - 600**  **Dolphins in pool OceanCurrent:**  **Spinny - 400**  **Dolphins in pool BlueAbyss:**  **none** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **Main**, **DolphinsPlay,** and **Dolphin**. **DolphinsPlay** class will have some methods, fields, and constructors. Cover the whole class with the unit test to make sure that the class is working as intended. In Judge, you upload **.zip** to **dolphinsPlay (**with **PlayDolphinsTests** inside**)** from the **skeleton**.