# Java OOP Retake Exam – 14 August 2024

1. **Overview**

You have to create a **forgottenBattleships** project, which keeps track of the remaining **battleships** after each battle conducted in a given **battle** zone. You can add different types of battleships in the battle zone. Your task is adding battleships, fighting battles and keeping statistics on them.

## Setup

* Upload **only the forgottenBattleships** 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 **2** interfaces and you must implement their functionalities in the **correct classes**.

The application has 2 types of entities: **BattleZone** and **Battleship**.

**Here is a UML diagram:**

A screenshot of a computer program

Description automatically generated

### BattleZone

BattleZoneImpl class holds information about the battleships and it should be instantiated.

#### Data

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

"**Battle zone name cannot be null or empty.**"

* + All names are unique.
* **capacity** - **int**
* **ships** - **Collection<Battleship>**

#### Behavior

##### Constructor

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

**(String name, int capacity)**

##### void addBattleship(Battleship battleship)

**Adds** a **Battleship** to the **BattleZone**.

* + If there is not enough capacity in the **BattleZone**, throw an **IllegalArgumentException** with the message:

**"Not enough capacity."**

* + If the **health** of the ship you are trying to add is below or equal to **0** (zero) throw an **IllegalArgumentException** with the message:

**"Ship's health cannot be below or equal to 0."**

##### Battleship getBattleshipByName(String battleshipName)

##### Parameters

* battleshipName - String

##### Functionality

Returns **BattleShip** with the given name.

##### void removeBattleship(Battleship battleship)

Removes a battleship from the collection.

##### Collection<Battleship> getShips()

Returns collection of ships in given **BattleZone**.

### Battleship

BaseBattleship is a **base class** of any **type of battleship** 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:

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

* **health - int** 
  + If the **health** drops below **0** (zero) **set** it to **0** (zero).
* **ammunition – int**
  + the **ammunition** amount of the ship. If it drops below **0** (zero) **set** it to **0** (zero).
* **hitStrength – int**
  + the amount of **damage** the ship inflicts.

#### Behavior

##### void attack(Battleship battleship)

* When the ship **attacks** its ammo amount **decreases**. Each ship implements this method with a different decreasing value.

##### void takeDamage(Battleship battleship)

* When the ship is **attacked** it takes **damage** equal to the **hit strength** of the **attacking** enemy ship.

#### Constructor

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

(String name, int health, int ammunition, int hitStrength)

#### Child Classes

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

##### RoyalBattleship

The constructorshould take the following values upon initialization:

**(String name, int health)**

Has **100** **initial units** of ammunition and **25 hit strength**.

#### Behavior

**void attack(Battleship battleship)**

* The method **reduces** the attacking **Battleship's** ammunition amount by **25 units**

**void takeDamage(Battleship battleship)**

* The method **reduces** the attacked **Battleship's** health by an amount of the **hit strength** of the enemy ship.

##### PirateBattleship

**Can only attack RoyalBattleship!**

The constructorshould take the following values upon initialization:

**(String name, int health)**

Has **80** **initial units** of ammunition and **10 hit strength**.

#### Behavior

**void attack(Battleship battleship)**

* The method **reduces** the attacking **Battleship's** ammunition amount by **10 units**

**void takeDamage(Battleship battleship)**

* The method **reduces** the attacked **Battleship's** health by an amount of the **hit strength** of the enemy ship.

## 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:

* **battleZones** - a **Collection** of **BattleZone**

### Commands

There are several **commands**, which control the **business** **logic** of the **application**. They are **stated** **below**. The **BattleZone** **name** passed to the methods will **always** be **valid**! You **must strictly follow** the order of checks in all methods as specified in this description!

#### AddBattleZone Command

##### Parameters

* battleZoneName- String
* capacity- int

##### Functionality

**Adds** a **BattleZone** to the Collection of battle zones.

If the **BattleZone** with the same name exists in the Collection, you have to **throw an IllegalArgumentException** with **the following message:**

* **"Battle zone exists!"**

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

* **"Successfully added {battle zone name} battle zone."**

#### GetBattleZoneByName Command

##### Parameters

* battleZoneName - String

##### Functionality

Returns **BattleZone** with the given name.

#### AddBattleshipToBattleZone Command

##### Parameters

* battleZoneName **–** String
* **shipType – String**
* **shipName – String**
* **health – int**

##### Functionality

**Adds** ship to the given battle zone. Valid battleships types are "**RoyalBattleship**" and "**PirateBattleship**".

If you are trying to add a ship to a non-existent battle zone, throw a **NullPointerException** with the message:

* **"Battle zone does not exist."**

If the **Battleship** **type** is **invalid**, throw an **IllegalArgumentException** with the message:

* **"Invalid ship type!"**

If a ship with the same name already exists in the given battlezone, throw an **IllegalArgumentException** with the message:

* **"This ship already exists!"**

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

* **"Successfully added {ship type} {ship name} to {battle zone name} battle zone."**

#### StartBattle Command

##### Parameters

* **battleZoneName** – **String**
* **attackingShip** – **String**
* **shipUnderAttack** – **String**

##### Functionality

When the start battle command is called, the action happens. Let’s take a quick look at the actions involved in the battle. For a successful battle, you need **two** battleships: an **attacker ship** and a **ship under attack**. The attacker ship **attacks** the ship under attack **once**, if it **has enough** units of ammunition.

When the attacker ship is attacking its ammunition decreases, and at this time, the ship under attack takes damage.

You have to implement the battle by following these steps:

* Get **battle zone** by its **name**. Keep in mind that it is guaranteed that the battle zone **will** **always** be **valid**.
* Before the battle begins you **have to verify** that you have **two battleships available** in the battle zone (**attacker** and **ship under attack**). If one of the battleships or both of them are **not available**, you **cannot** process the battle.

In that case, you have to throw an **IllegalArgumentException** with the message**:**

* **"You need at least two battleships added to the battle zone to continue."**
* If **you have two available battleships** in the battle zone (attacker ship and ship under attack), you **can begin** the battle between the two ships. The **attacker** ship attacks the ship under attack ship **once** if it has **enough units of ammunition**. At the same time, the **ship under attack** **takes damage**.
* When the battle is over, you have to check if one of the ships is **with health value** that is **equals or lower** than **zero**. In that case, you **have to remove** the ship from the battle zone.
* Finally, **return** information about the remaining ships in the battle with the following message:
* **"The battle in {battle zone name} continues. Ships still in battle: shipName1, shipName2, …shipNamen"**

#### GetStatistics Command

##### Functionality

Returns information about each **BattleZone**.

If there are **two or more** battleships in the battle zone print the following:

**"Ships in {battle zone name} battle zone:**

**-- {ship name} - health: {ship health}, ammunition: {ship ammunition}**

**-- {ship name} - health: {ship health}, ammunition: {ship ammunition}**

**-- {ship name} - health: {ship health}, ammunition: {ship ammunition}**

**(…)"**

If there is **only one** battleship remaining in the battle zone print the following:

**"Ships in {battle zone name} battle zone:**

**{ship name} is the only one who survived and won the battle!"**

There will always be **at least** one ship that wins the battle.

It is recommended to use ***System.lineSeparator()*** instead of **"\n"** when appending a new line when printing the result of this method.

#### 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.

**You must implement all the methods in EngineImpl!**

#### Input

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

* **AaddBattleZone(String battleZoneName, int capacity)**
* **GetBattleZoneByName(String battleZoneName)**
* **AddBattleShipToBattleZone(String battleZoneName, String shipType, String shipName, int health)**
* **StartBattle(String battleZoneName, String attackingShip, String shipUnderAttack)**
* **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** |
| **AddBattleZone BlackSea 5**  **AddBattleshipToBattleZone BlackSea RoyalBattleship RoyalGlory 20**  **AddBattleshipToBattleZone BlackSea RoyalBattleship QueenVictoria 70**  **AddBattleshipToBattleZone BlackSea PirateBattleship FlyingDutchman 50**  **StartBattle BlackSea RoyalGlory QueenVictoria**  **StartBattle BlackSea RoyalGlory QueenVictoria**  **StartBattle BlackSea RoyalGlory FlyingDutchman**  **StartBattle BlackSea QueenVictoria FlyingDutchman**  **StartBattle BlackSea RoyalGlory QueenVictoria**  **AddBattleshipToBattleZone BlackSea PirateBattleship DyingGull 30**  **StartBattle BlackSea RoyalGlory DyingGull**  **StartBattle BlackSea DyingGull RoyalGlory**  **StartBattle BlackSea DyingGull RoyalGlory**  **StartBattle BlackSea DyingGull RoyalGlory**  **GetStatistics**  **Exit** |
| **Output** |
| **Successfully added BlackSea battle zone.**  **Successfully added RoyalBattleship RoyalGlory to BlackSea battle zone.**  **Successfully added RoyalBattleship QueenVictoria to BlackSea battle zone.**  **Successfully added PirateBattleship FlyingDutchman to BlackSea battle zone.**  **The battle in BlackSea continues. Ships still in battle: RoyalGlory, QueenVictoria, FlyingDutchman**  **The battle in BlackSea continues. Ships still in battle: RoyalGlory, QueenVictoria, FlyingDutchman**  **The battle in BlackSea continues. Ships still in battle: RoyalGlory, QueenVictoria, FlyingDutchman**  **The battle in BlackSea continues. Ships still in battle: RoyalGlory, QueenVictoria**  **The battle in BlackSea continues. Ships still in battle: RoyalGlory**  **Successfully added PirateBattleship DyingGull to BlackSea battle zone.**  **The battle in BlackSea continues. Ships still in battle: RoyalGlory, DyingGull**  **The battle in BlackSea continues. Ships still in battle: RoyalGlory, DyingGull**  **The battle in BlackSea continues. Ships still in battle: DyingGull**  **You need at least two battleships added to the battle zone to continue.**  **Ships in BlackSea battle zone:**  **DyingGull is the only one who survived and won the battle!** |
| **Input** |
| **AddBattleZone Atlantic 3**  **AddBattleshipToBattleZone Atlantic RoyalBattleship RoyalGlory 20**  **AddBattleshipToBattleZone Atlantic RoyalBattleship QueenVictoria 20**  **AddBattleshipToBattleZone Atlantic PirateBattleship BlackPearl -50**  **AddBattleshipToBattleZone Atlantic PirateBattleship FlyingSkeleton 50**  **AddBattleshipToBattleZone Atlantic PirateBattleship BlackPearl 50**  **StartBattle Atlantic RoyalGlory QueenVictoria**  **StartBattle Atlantic FlyingSkeleton RoyalGlory**  **GetStatistics**  **Exit** |
| **Output** |
| **Successfully added Atlantic battle zone.**  **Successfully added RoyalBattleship RoyalGlory to Atlantic battle zone.**  **Successfully added RoyalBattleship QueenVictoria to Atlantic battle zone.**  **Ship's health cannot be below or equal to 0.**  **Successfully added PirateBattleship FlyingSkeleton to Atlantic battle zone.**  **Not enough capacity.**  **The battle in Atlantic continues. Ships still in battle: RoyalGlory, FlyingSkeleton**  **The battle in Atlantic continues. Ships still in battle: RoyalGlory, FlyingSkeleton**  **Ships in Atlantic battle zone:**  **-- RoyalGlory - health: 10, ammunition: 75**  **-- FlyingSkeleton - health: 50, ammunition: 70** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **Main**, **Ship,** and **Battle**. **Battle** 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 **battleShip (**with **BattleTests** inside**)** from the **skeleton**.

**Note that the Unit tests use a new version of JUnit -> Jupiter and there are a few syntax changes and a small change in error handling. Please take a look at the following:**

**Import from org.junit.jupiter.api.\***

* import **org.junit.jupiter.api.BeforeEach;**
* import **org.junit.jupiter.api.Test;**
* import **org.junit.jupiter.api.Assertions;**
* **assert** is replaced by **Assertions**
* **@Before** annotation is replaced by **@BeforeEach**

Handling of exceptions example:

With JUnit 4:

@Test**(expected = NullPointerException.class)**

public void addCar() {

car = null;

carShop.add(car);

}

With JUnit Jupiter:

@Test

Public void addCar() {

**Assertions.assertThrows(NullPointerException.class, () -> {**

car = null;

carShop.add(car);

**});**

**This is just an example and it is not related to any task of this exam!**