# Java OOP Retake Exam – 19 December 2022



## Overview

In this exam, your task will be to create a basic magic game. In the game, there are **Region**, **Маgicians** of different teams, and **Magic**.

## Setup

* Upload **only the** magicGamepackage 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 interfaces, and you have to implement their functionality in the **correct classes**.

There are **3** types of entities in the application: **Magic**, **Magicians**, **and Region**. There should also be **MagicRepositoryImpl** and **MagicianRepositoryImpl**.

### MagicImpl

MagicImpl is a **base class** of any **type of magic** 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: **"Magic cannot be null or empty."**
  + All names are unique.
* **bulletsCount – int**
  + If the bullets count is below zero**,** throw an **IllegalArgumentException** with a message: **"Bullets cannot be below 0."**

#### Behavior

##### int fire()

The **fire()** method fires the number of bullets the **Magic** has and reduces its available amount. **RedMagic** can fire only 1 bullet and the **BlackMagic** only 10 at once, **not more, not less**. If there are **not enough** bullets, the method should return 0.

#### Constructor

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

**(String name, int bulletsCount)**

#### Child Classes

There are two types of **Magic**:

##### RedMagic

The constructorshould take the following values upon initialization:

(**String name, int bulletsCount**)

**RedMagic can fire() 1 bullet at a time.**

##### BlackMagic

The constructorshould take the following values upon initialization:

(**String name, int bulletsCount**)

**BlackMagic can fire() 10 bullets at a time.**

### MagicianImpl

MagicianImpl is a **base class** for any **type of magician,** and it **should not be able to be instantiated**.

#### Data

* **username** - **String**
  + If the username **is null or whitespace,** throw a **NullPointerException** with a message: **"Username cannot be null or empty."**
  + All names are unique.
* **health** - **int**
  + If the health is below 0**,** throw an **IllegalArgumentException** with a message: **"Magician health cannot be below 0."**
* **protection** - **int**
  + If the protection is below 0**,** throw an **IllegalArgumentException** with a message: **"Magician protection cannot be below 0."**
* **isAlive** - **boolean**
  + If the health is above zero.
* **magic** - **Magic**
  + If the magic object is null**,** throw a **NullPointerException** with a message: **"Magic cannot be null."**

#### Behavior

##### void takeDamage(int points)

The **takeDamage()** method decreases the **Magician's** protection and health. First, you need to reduce the protection. If the protection reaches 0, transfer the damage to health points. If the health points are less than or equal to zero, the magician is dead.

#### Constructor

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

**(String username, int health, int protection, Magic magic)**

#### Child Classes

There are two types of **Magician**:

##### Wizard

The constructorshould take the following values upon initialization:

**(String username, int health, int protection, Magic magic)**

##### BlackWidow

The constructorshould take the following values upon initialization:

**(String username, int health, int protection, Magic magic)**

### RegionImpl

The **RegionImpl** is the ground where the wizard fights happen.

#### Behavior

##### String start(Collection<Magician> magicians)

Separates the magicians into two types - **Wizard** and **Black Widow**. The game continues until one of the teams is completely dead (all magicians have 0 health). Both magician groups take turn shooting at each other – first are the **Wizards**, then the **Black Widows** inflict damage equal to their bullet fired from their **Magic**. Make sure the **Magician** has enough bullets before he/she tries to attack!

The damage they deal comes from each magic property of each Magician.

If Wizards win **return** "**Wizards win!"** otherwise **return** "**Black widows win!**"

### MagicRepositoryImpl

The **magic repository** is a **repository** for all **magics** in the game.

#### Data

* data - **a** **Collection of magics**

#### Behavior

##### void addMagic(Magic magic)

* If the magic is null**,** throw a **NullPointerException** with a message: **"Cannot add null in Magic Repository."**
* **Adds** a **magic** to the **collection**.

##### boolean removeMagic(Magic magic)

* **Removes** a **magic** from the **collection**. **Returns true** if the removal was **successful**, **otherwise** - **false**.

**Magic findByName(String name)**

* **Returns** the **first** **magic** with the **given name**, if there is such a magic. **Otherwise**, returns **null**.

### MagicianRepositoryImpl

The **magician repository** is a **repository** for all **magicians** in the game.

#### Data

* data - **a** **collection of magicians**

#### Behavior

##### void addMagician(Magician magician)

* If the magician is null**,** throw a **NullPointerException** with a message: "**Cannot add null in Magician Repository."**.
* **Adds** a **magician** to the **collection**.

##### boolean removeMagician(Magician magician)

* **Removes** a **magician** from the **collection**. **Returns true** if the removal was **successful**, **otherwise** - **false**.

**Magician findByName(String name)**

* **Returns** the **first** **magician** with the **given username**, if there is such a magician. **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 that you must 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 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:

* **magics** - **MagicRepositoryImpl**
* **magicians** – **MagicianRepositoryImpl**
* **region - Region**

### Commands

There are several **commands**, which control the **business** **logic** of the **application**. They are **stated** **below**.

#### AddMagic Command

##### Parameters

* type - String
* name - String
* bulletsCount - int

##### Functionality

**Adds** a **Magic** and **adds** it to the **MagicRepositoryImpl**. **Valid** types are "**RedMagic**" and "**BlackMagic**".

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

* **"Invalid magic type."**

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

* **"Successfully added magic {magicName}."**

#### AddMagician Command

##### Parameters

* type – String
* username – String
* health – int
* protection – int
* magicName – String

##### Functionality

**Creates** a **Magician** of the **given type** and **adds** it to the **MagicianRepositoryImpl**. **Valid** types are: "**Wizard**" and "**BlackWidow**".

If the **magic** is **not found** throw **NullPointerException** with a message:

* **"Magic cannot be found!"**

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

* **"Invalid magician type!"**

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

* **"Successfully added magician {username}."**

#### StartGame Command

##### Functionality

The game starts with all magicians that are **alive**! Returns the result from the **start()** method.

#### Report Command

##### Functionality

Returns information about each magician separated with a new line. Order then by health descending, then by username alphabetically, them by type alphabetically. You can use the overridden **.toString() Magician** method.

**"{magician type}: {magician username}**

**Health: {magician health}**

**Protection: {magician protection}**

**Magic: {magician magic name}"**

**Note: Use System.lineSeparator() for a new line and don't forget to trim if you use StringBuilder.**

### Input / Output

You are provided with one interface, which will help 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. The **Main** method is **complete**. Do not add anything to it.

#### Input

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

* **AddMagic** **{type} {name} {bulletsCount}**
* **AddMagician {type} {username} {health} {protection} {magicName}**
* **StartGame**
* **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** |
| **AddMagic RedMagic Calypso 100**  **AddMagic RedMagic Faye 100**  **AddMagic Granate Invalid 100**  **AddMagic BlackMagic Sabrina** **5**  **AddMagic BlackMagic Luna 15**  **AddMagician Wizard bestWizz 50 50 Calypso**  **AddMagician Wizard topOfTheTop 50 50 Faye**  **AddMagician Wizard javaLover 20 -50 Faye**  **AddMagician BlackWidow JohnLoveJava 50 50 Sabrina**  **AddMagician BlackWidow magicianPeter 30 30 Luna**  **AddMagician Player Invalid 30 30 Luna**  **StartGame**  **Report**  **Exit** |
| **Output** |
| **Successfully added magic Calypso.**  **Successfully added magic Faye.**  **Invalid magic type!**  **Successfully added magic Sabrina.**  **Successfully added magic Luna.**  **Successfully added magician bestWizz.**  **Successfully added magician topOfTheTop.**  **Magician protection cannot be below 0.**  **Successfully added magician JohnLoveJava.**  **Successfully added magician magicianPeter.**  **Invalid magician type!**  **Wizards win!**  **BlackWidow: JohnLoveJava**  **Health: 0**  **Protection: 0**  **Magic: Sabrina**  **BlackWidow: magicianPeter**  **Health: 0**  **Protection: 0**  **Magic: Luna**  **Wizard: bestWizz**  **Health: 50**  **Protection: 40**  **Magic: Calypso**  **Wizard: topOfTheTop**  **Health: 50**  **Protection: 50**  **Magic: Faye** |

|  |
| --- |
| **Input** |
| **AddMagic RedMagic Calypso 100**  **AddMagic RedMagic Faye 100**  **AddMagic BlackMagic Luna 15**  **AddMagic BlackMagic Iris 25**  **AddMagician Wizard bestWizz 50 44 Calypso**  **AddMagician Wizard topOfTheTop 50 0 Faye**  **AddMagician BlackWidow AlexJs 100 100 Iris**  **AddMagician BlackWidow WidowSara 100 100 Sabrina**  **StartGame**  **Report**  **Exit** |
| **Output** |
| **Successfully added magic Calypso.**  **Successfully added magic Faye.**  **Successfully added magic Luna.**  **Successfully added magic Iris.**  **Successfully added magician bestWizz.**  **Successfully added magician topOfTheTop.**  **Successfully added magician AlexJs.**  **Magic cannot be found!**  **Wizards win!**  **BlackWidow: AlexJs**  **Health: 0**  **Protection: 0**  **Magic: Iris**  **Wizard: topOfTheTop**  **Health: 40**  **Protection: 0**  **Magic: Faye**  **Wizard: bestWizz**  **Health: 50**  **Protection: 34**  **Magic: Calypso** |

# Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **Main**, **Magic,** and **Magician**. **Magician** 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 **magicGame (**with **MagicianTests** inside**)** from the **skeleton**.

The imports you will need are **org.junit.Assert** and **org.junit.Test**. You may also have additional ones that are default (for example **java.unit.List**).