Java OOP Exam - 12 April 2020



1. Overview

In this exam your task will be to create a basic Shooter game. In the game there are Field, Players of different teams and Guns.

2. Setup

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

There are 3 types of entities in the application: Gun, Player, Field. There should also be GunRepository and PlayerRepository.

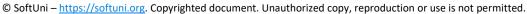
GunImpl

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

Data

- name String
 - If the name is null or whitespace, throw a NullPointerException with message: "Gun cannot be null or empty."
 - All names are unique
- **bulletsCount** int
 - If the bullets count is below zero, throw an IllegalArgumentException with message: "Bullets cannot be below 0."





















Behavior

int fire()

The fire() method returns the number of bullets fired. Pistol can fire only 1 bullet and the Rifle only 10 at once, not more, not less. If there are not enough bullets, the method should return 0.

Constructor

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

(String name, int bulletsCount)

Child Classes

There are two types of **Gun**:

Pistol

Constructor should take the following values upon initialization:

(String name, int bulletsCount)

Pistol can fire() 1 bullet at a time.

Rifle

Constructor should take the following values upon initialization:

(String name, int bulletsCount)

Rifle can fire() 10 bullets at a time.

PlayerImpl

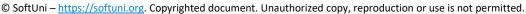
Player is a base class for any type of player, and it should not be able to be instantiated.

Data

- username-String
 - o If the username is null or whitespace, throw a NullPointerException with message: "Username cannot be null or empty."
 - All names are unique
- health int
 - If the health is below 0, throw an IllegalArgumentException with message: "Player health cannot be below 0."
- armor int
 - If the armor is below 0, throw an IllegalArgumentException with message: "Player armor cannot be below 0."
- isAlive boolean
 - If the health is above zero
- gun Gun
 - o If the gun is null, throw a **NullPointerException** with message:

"Gun cannot be null."























Behavior

void takeDamage(int points)

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

Constructor

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

(String username, int health, int armor, Gun gun)

Child Classes

There are two types of **Player**:

Terrorist

Constructor should take the following values upon initialization:

(String username, int health, int armor, Gun gun)

CounterTerrorist

Constructor should take the following values upon initialization:

(String username, int health, int armor, Gun gun)

FieldImpl

Behavior

String start(Collection<Player> players)

Separates the players in two types - Terrorist and Counter Terrorist. The game continues until one of the teams is completely dead (all players have 0 health). The terrorists attack first and after that the counter terrorists. The attack happens like that: Each live terrorist shoots on each live counter terrorist once and inflicts damage equal to the bullets fired and after that each live counter terrorist shoots on each live terrorist.

If Terrorists win returns "Terrorist wins!" otherwise returns "Counter Terrorist wins!"

GunRepository

The **gun repository** is a **repository** for all **guns** in the game.

Data

models - a collection of guns

Behavior

void add(Gun gun)

- If the gun is null, throw a NullPointerException with message: "Cannot add null in Gun Repository".
- Adds a gun in the collection.



















boolean remove(Gun gun)

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

Gun findByName(String name)

Returns the first gun with the given name, if there is such gun. Otherwise, returns null.

PlayerRepository

The player repository is a repository for all players in the game.

Data

models - a collection of players

Behavior

void add(Player player)

- If the player is null, throw a NullPointerException with message: "Cannot add null in Player Repository".
- Adds a player in the collection.

boolean remove(Player player)

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

Player findByName(String name)

Returns the first player with the given username, if there is such player. 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 Controller 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 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.

Data

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

- guns GunRepository
- players PlayerRepository
- field Field

Commands

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



















AddGun Command

Parameters

- type String
- name String
- bulletsCount int

Functionality

Adds a Gun and adds it to the GunRepository. Valid types are: "Pistol" and "Rifle".

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

"Invalid gun type."

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

"Successfully added gun {gunName}."

AddPlayer Command

Parameters

- type-String
- username String
- health int
- armor int
- gunName String

Functionality

Creates a Player of the given type and adds it to the PlayerRepository. Valid types are: "Terrorist" and "CounterTerrorist".

If the **gun** is **not found** throw **NullPointerException** with message:

"Gun cannot be found!"

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

"Invalid player type!"

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

"Successfully added player {playerUsername}."

StartGame Command

Functionality

Game starts with all players that are alive! Returns the result from the start() method.

Report Command

Functionality

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

```
"{player type}: {player username}
--Health: {player health}
```





















```
--Armor: {player armor}
--Gun: {player gun 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 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 are given the EngineImpl class with written logic in it. In order for the code to be compiled, some parts are commented, don't forget to uncomment them.

Input

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

- AddGun {type} {name} {bulletsCount}
- AddPlayer {type} {username} {health} {armor} {gunName}
- 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
AddGun Rifle Express 100
AddGun Rifle Buffalo 100
AddGun Rifle Assault 100
AddGun Granate Invalid 100
AddGun Pistol Kolibri 5
AddGun Pistol Makarov 15
AddGun Pistol Magnum 3
AddGun Pistol 3
AddPlayer Terrorist Shopoff 50 50 Express
AddPlayer Terrorist Kris 50 50 Buffalo
AddPlayer Terrorist 50 50 Express
AddPlayer Terrorist Atanas 50 50 Invalid
AddPlayer Terrorist Atanas -10 50 Express
AddPlayer Terrorist Atanas 20 -50 Express
AddPlayer CounterTerrorist John 50 50 Kolibri
AddPlayer CounterTerrorist Peter 30 30 Makarov
AddPlayer Player Invalid 30 30 Makarov
StartGame
Report
Exit
Output
```



Successfully added gun Express. Successfully added gun Buffalo. Successfully added gun Assault.

















Invalid gun type! Successfully added gun Kolibri. Successfully added gun Makarov. Successfully added gun Magnum. Gun cannot be null or empty. Successfully added player Shopoff. Successfully added player Kris. Username cannot be null or empty. Gun cannot be found! Player health cannot be below 0. Player armor cannot be below 0. Successfully added player John. Successfully added player Peter. Invalid player type! Terrorist wins! CounterTerrorist: John --Health: 0 --Armor: 0 --Gun: Kolibri CounterTerrorist: Peter --Health: 0 --Armor: 0 --Gun: Makarov Terrorist: Kris --Health: 50 --Armor: 46 --Gun: Buffalo Terrorist: Shopoff

Input

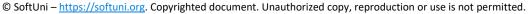
--Health: 50 --Armor: 45 --Gun: Express

AddGun Rifle Express 1000 AddGun Rifle Buffalo 1000 AddGun Rifle Assault 1000 AddGun Pistol Kolibri 20 AddGun Pistol Makarov 20 AddGun Pistol Magnum 20 AddPlayer Terrorist Shopoff 50 44 Makarov AddPlayer Terrorist Kris 50 0 Magnum AddPlayer Terrorist Atanas 50 10 Kolibri AddPlayer CounterTerrorist John 100 100 Express AddPlayer CounterTerrorist Peter 100 100 Buffalo **StartGame** Report Exit

Output

Successfully added gun Express. Successfully added gun Buffalo. Successfully added gun Assault. Successfully added gun Kolibri. Successfully added gun Makarov. Successfully added gun Magnum. Successfully added player Shopoff. Successfully added player Kris. Successfully added player Atanas.





















Successfully added player John. Successfully added player Peter.

Counter Terrorist wins!
CounterTerrorist: John

--Health: 100 --Armor: 89 --Gun: Express

CounterTerrorist: Peter

--Health: 100 --Armor: 89 --Gun: Buffalo Terrorist: Atanas

--Health: 0
--Armor: 0
--Gun: Kolibri
Terrorist: Kris
--Health: 0
--Armor: 0
--Gun: Magnum
Terrorist: Shopoff

--Health: 0 --Armor: 0 --Gun: Makarov

Task 3: Unit Tests (100 points)

You will receive a skeleton with **Gun** and **Player** classes inside. The class will have some methods, fields and one constructor, which are working properly. You are **NOT ALLOWED** to change any classes. 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!

















