Java OOP Retake Exam - 15 August 2019

1. Overview

Space Missions are very interesting and you have been sent to such one. Your mission is create a **Space Station** project, which navigates astronauts missions for collecting items from a foreign planet. The Space Station has **Astronauts** with different professional speacialties and their capability to survive in open space differs according to their essential needs, like the need for oxygen. Your task is to send them on missions and collect items from the different planets.

2. Setup

- Upload **only the `spaceStation`** package in every problem **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 **5** interfaces, and you have to implement their functionality in the **correct classes**.

There are **5** types of entities in the application: **`Astronaut`, `Bag`, `Mission`, `Planet`**. There should also have be a **`AstronautRepository`**, as well as **`PlanetRepository`**.

BaseAstronaut

`BaseAstronaut` is a **base class** or any **type of astronaut** and it **should not be able to be instantiated**.

Data

- **`name`** – **`String`**
    - If the name **is null or whitespace,** throw a **`NullPointerException`** with message: **"Astronaut name cannot be null or empty."**
    - All names are unique
- **`oxygen`** – **`double`**
    - The oxygen of an astronaut
    - If the oxygen is below **0,** throw an **`IllegalArgumentException`** with message: **`"Cannot create Astronaut with negative oxygen!"`**
- **`bag`** – **`Bag`**
    - A field of type **`Backpack`**

Behavior

`void breath()`

The **`breath()`** method decreases astronauts' oxygen. Keep in mind that some types of Astronaut can implement the method in a different way.

- The method **decreases** the astronauts' oxygen by **10 units**.

- Astronaut's oxygen should not drop below zero

Constructor

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

`String name, double oxygen`

Child Classes

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

Biologist

Has **70 initial units of oxygen**.

The method **`breath()` decreases** the astronauts' oxygen by **5 units**.

Constructor should take the following values upon initialization:

`String name`

Geodesist

Has **50 initial units of ogyxen**.

Constructor should take the following values upon initialization:

`String name`

Meteorologist

Has **initial 90 units of oxygen.**

Constructor should take the following values upon initialization:

`String name`

Backpack

The **`Backpack`** is class that holds collection of items. **It should** be able to be **instantiated**.

Data

- **`items –`** a collection of **Strings**

Constructor

The constructor should not take any values upon initialization.

PlanetImpl

The **`PlanetImpl`** is a class that holds information about the items that can be found on its surface. **It should** be able to be **instantiated**.

Data

- **`name – String`**
  - If the name **is null or whitespace,** throw a **`NullPointerException`** with message: **"Invalid name!"**
- **`items`** – a collection of Strings

Constructor

The constructor should take the following values upon initialization:

`String name`

MissionImpl

The **`MissionImpl`** class holds the main action, which is the **`explore`** method.

Behavior

`void explore(Planet planet, Collection<Astronaut> astronauts)`

Here is how the **`Explore`** method works:

- The astronauts start going out in open space one by one. They **can't go**, if they don't have **any oxygen** left.
- An astronaut lands on a planet and **starts collecting its items one by one**.
- He **finds an item** and he **takes a breath**.
- He **adds the item** to his **backpack** and respectively the item **must be removed** from the planet.
- **Astronauts can't keep collecting items** if their **oxygen becomes 0**.
- **If it becomes 0, the next astronaut starts exploring.**

AstronautRepository

The astronaut repository is a repository for the astronauts that are on the Space Station.

Data

- **`astronauts`** – **a collection of astronauts**

Behavior

`void add(Astronaut astronaut)`

- Adds an astronaut in the Space Station.
- Every astronaut is unique and it is guaranteed that there will not be an astronaut with the same name.

`boolean remove(Astronaut astronaut)`

- Removes an astronaut from the collection. Returns true if the deletion was sucessful.

`Astronaut findByName(String name)`

- Returns an astronaut with that name.

`Collection<Astronaut> getModels()`

- Returns collection of astronauts (unmodifiable)

PlanetRepository

The planet repository is a repository for planets that await to be explored.

Data

- **`planets`** – **a collection of planets**

Behavior

`void add(Planet planet)`

- Adds a planet for exploration.
- Every planet is unique and it is guaranteed that there will not be a planet with the same name.

`boolean remove(Planet planet)`

- Removes a planet from the collection. Returns true if the deletion was sucessful.

`Planet findByName(String name)`

- Returns a planet with that name.
- It is guaranteed that the planet exists in the collection.

`Collection<Planet> getModels()`

- Returns collection of planets (unmodifiable)

4. 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 of its methods. The constructor of **`ControllerImpl`** does not take any arguments. The given methods should have the following logic:

Commands

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

AddAstronaut Command

Parameters

- **`type – String`**
- **`astronautName - String`**

Functionality

Creates an astronaut with the given name of the given type. If the astronaut is invalid, throw an **`IllegalArgumentException`** with message:

"Astronaut type doesn't exists!"

The method should **return** the following message:

- **"Successfully added {astronautType}: {astronautName}!"**

AddPlanet Command

Parameters

- **`planetName - String`**
- **`items – String...`**

Functionality

Creates a **planet** with the provided **items** and **name**.

The method should **return** the following message:

- **"Successfully added Planet: {planetName}!"**

RetireAstronaut Command

Parameters

- `astronautName – String`

Functionality

Retires the astronaut from the space station by removing it from its repository. If an astronaut with that name doesn't exist, **throw `IllegalArgumentException`** with the following message:

- **"Astronaut {astronautName} doesn't exists!"**

If an astronaut is successfully retired, **remove it from the repository** and **return** the following **message**:

- **"Astronaut {astronautName} was retired!"**

ExplorePlanet Command

Parameters

- `planetName - String`

Functionality

When the explore command is called, the action happens. You should start exploring the given planet, by sending the astronauts that are most suitable for the mission:

- You call each of the astronauts and pick only the ones that have oxygen above 60 units.
- You send the suitable astronauts on a mission to explore the planet.
- If you **don't have any suitable astronauts**, throw **`IllegalArgumentException`** with the following message: **"You need at least one astronaut to explore the planet!"**
- After a mission, you must **return the following message**, with the **name of the explored planet** and the **count** of the **astronauts** that **had given their lives** for the mission:
  "Planet: {planetName} was explored! Exploration finished with {deadAstronauts} dead astronauts!"

Report Command

Functionality

Returns the information about the astronauts. If any of them **doesn't have bag items**, print **"none"** instead:

"{exploredPlanetsCount} planets were explored!"

"Astronauts info:"

"Name: {astronautName One}"

"Oxygen: {astronautOxygen One}"

"Bag items: {bagItem$_1$, bagItem$_2$, bagItem$_3$, …, bagItem$_n$ \ "none"}"

…

"Name: {astronautName N}"

"Oxygen: {astronautOxygen N}"

"Bag items: {bagItem$_1$, bagItem$_2$, bagItem$_3$, …, bagItem$_n$ \ "none"}"

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:

- **AddAstronaut {astronautType} {astronautName}**
- **AddPlanet {planetName} {item$_1$} {item$_2$}… {item$_N$}**
- **RetireAstronaut {astronautName}**
- **ExplorePlanet {planetName}**
- **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
AddAstronaut Biologist Oliver AddAstronaut Geodesist Jake AddAstronaut Meteorologist James AddAstronaut Biologist Michael AddAstronaut Meteorologist David AddAstronaut Meteorologist Thomas AddAstronaut Engineer Alexander AddPlanet Mercury AddPlanet Mars Carbon RetireAstronaut David ExplorePlanet Mars Report Exit
Output
Successfully added Biologist: Oliver! Successfully added Geodesist: Jake! Successfully added Meteorologist: James! Successfully added Biologist: Michael! Successfully added Meteorologist: David! Successfully added Meteorologist: Thomas! Astronaut type doesn't exists! Successfully added Planet: Mercury! Successfully added Planet: Mars! Astronaut David was retired! Planet: Mars was explored! Exploration finished with 0 dead astronauts! 1 planets were explored! Astronauts info: Name: Oliver Oxygen: 65 Bag items: Carbon Name: Jake Oxygen: 50 Bag items: none

```
Name: James
Oxygen: 90
Bag items: none
Name: Michael
Oxygen: 70
Bag items: none
Name: Thomas
Oxygen: 90
Bag items: none
```

Input

```
AddAstronaut Geodesist Jake
AddPlanet Mars Carbon
ExplorePlanet Mars
Report
AddAstronaut Biologist Jack
AddAstronaut Meteorologist Liam
AddAstronaut Biologist Michael
AddAstronaut Meteorologist David
AddAstronaut Meteorologist Thomas
AddPlanet Jupiter Titanium Quartz Aluminium Azurnium Cobalt Copper Iron Lead Lithium Plutonium Mercury Nickel Magnesium Diamond Gold Carbon
RetireAstronaut David
RetireAstronaut William
ExplorePlanet Jupiter
Report
Exit
```

Output

```
Successfully added Geodesist: Jake!
Successfully added Planet: Mars!
You need at least one astronaut to explore the planet!
0 planets were explored!
Astronauts info:
Name: Jake
Oxygen: 50
Bag items: none
Successfully added Biologist: Jack!
Successfully added Meteorologist: Liam!
Successfully added Biologist: Michael!
Successfully added Meteorologist: David!
Successfully added Meteorologist: Thomas!
Successfully added Planet: Jupiter!
Astronaut David was retired!
Astronaut William doesn't exists!
Planet: Jupiter was explored! Exploration finished with 1 dead astronauts!
1 planets were explored!
Astronauts info:
Name: Jake
Oxygen: 50
Bag items: none
Name: Jack
Oxygen: 0
Bag items: Titanium, Quartz, Aluminium, Azurnium, Cobalt, Copper, Iron, Lead, Lithium, Plutonium, Mercury, Nickel, Magnesium, Diamond
Name: Liam
Oxygen: 70
Bag items: Gold, Carbon
Name: Michael
```

```
Oxygen: 70
Bag items: none
Name: Thomas
Oxygen: 90
Bag items: none
```

5. Task 3: Unit Tests (100 points)

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