Java OOP Retake Exam – 22.08.2021

1. Overview

The missions in Antarctica are very interesting and you have been chosen to go on a special one. Your mission is to design a station in Antarctica to navigate the missions of polar explorers. The station has researchers with different professional specialties and their ability to survive the cold varies depending on their basic needs, such as the need for a variety of food and heat. Your task is to send them on missions and collect exhibits from the different expeditions.

2. Setup

- Upload a zip containing only the **glacialExpedition** package as a **solution** to every problem except for the Unit Testing problem.
- **Do not modify the provided 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 implementational structure by adding more public methods in the classes than the interfaces have defined**
- Make sure you have **no public fields** anywhere

3. Task 1: Structure (50 points)

You are given **4** interfaces and you must implement their functionalities in the **correct classes**.

There are **4** types of entities in the application: **`Explorer`, `Suitcase`, `Mission`, `State`.** There are also 2 repositories: an **`ExplorerRepository`** and a **`StateRepository`**.

Explorer

`BaseExplorer` is a **base class** or any **type of explorer** and **should not be instantiated**.

Data

- **`name`** – **`String`**
    - If the value of the name is either **null** or **empty** (containing only whitespaces), throw a **NullPointerException** with the following message: **"Explorer name cannot be null or empty."**
    - The values of the names are **unique.**

- **`energy – double`**
  - The energy of an explorer
  - If the energy is a **negative** number, throw an **IllegalArgumentException** with the following message: **"Cannot create Explorer with negative energy."**
- **`suitcase – Suitcase`**
  - **`A Suitcase field type`**

Behavior

`void search()`

The **search()** method decreases the explorer's energy. Keep in mind that some Explorer types can implement the method in a different way.

- The method **decreases** the explorer's energy by **15 units**.
- The energy value **should not** drop **below zero**.
- Set the value to be zero, if the energy value goes below zero.

`void canSearch()`

The **`canSearch()`** method returns **boolean**. Tell us if the energy more than zero.

Constructor

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

`String name, double energy`

Child Classes

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

NaturalExplorer

Has **60 initial units of energy**.

The **search()** method should **decrease** the explorer's energy by **7** units.

The constructor should take the following values upon an initialization:

`String name`

GlacierExplorer

Has **40 initial units of energy**.

The constructor should take the following values upon an initialization:

`String name`

AnimalExplorer

Has **initial 100 units of energy.**

The constructor should take the following values upon an initialization:

`String name`

Suitcase

The **`Carton`** class holds a **collection** of **exhibits**. It should be **instantiated**.

Data

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

Constructor

The constructor should not take any values upon an initialization.

State

The **StateImpl** class holds the information about the **exhibits** that can be **found** and **explored**. It should be instantiated.

Data

- **`name - String`**
  - If the value of the **name** is either **null** or **empty** (containing only whitespaces), throw a **NullPointerException** with the following message: **"Invalid name!"**
- **`exhibits`** – a collection of Strings

Constructor

The constructor should take the following values upon initialization:

`String name`

Mission

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

Behavior

```
void explore(State state, Collection<Explorer> explorers)
```

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

- Explorers **cannot** go on expeditions if their **energy** is **below** 0.
- They leave the station and **start collecting exhibits** one by one.
- If they **find** an exhibit, their **energy** is **decreased**.
- They add the **exhibit** to their **carton**. The **exhibit** should then be **removed** from the **state**.
- Explorers **cannot continue** collecting exhibits if their **energy drops** to 0.
  - If their energy drops to 0, the next explorer starts exploring**.**

ExplorerRepository

The **ExplorerRepository** class is a **repository** for the **explorers**.

Data

- **`explorers`** – **a collection of explorers**

Behavior

```
void add(Explorer explorer)
```

- Adds an explorer to the station.
- Every explorer is unique in the collection.
  - It is guaranteed that there will not be an explorer with the same name.

`boolean remove(Explorer explorer)`

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

`Explorer byName(String name)`

- Returns an explorer with that name.
- If the explorer is not in the collection, return null.

`Collection<Explorer> getCollection()`

- Returns an unmodifiable collection of explorers.

StateRepository

The **StateRepository** class is a **repository** for the **unexplored states**.

Data

- **`states` – a collection of states**

Behavior

`void add(State state)`

- Adds a state for exploration.
- Every state is unique in the collection.
    - It is guaranteed that there will not be a state with the same name.

`boolean remove(State state)`

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

`State byName(String name)`

- Returns a state with that name.
- If the state is not in the collection, return null.

`Collection<State> getCollection()`

- Returns an unmodifiable collection of states.

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 must implement in the correct classes.

Note: The `ControllerImpl` class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!

The **`Controller`** interface is the first interface. You must create a **`ControllerImpl`** class, which implements the interface and implements all its methods. The **`ControllerImpl's`** constructor 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.

AddExplorer Command

Parameters

- `type – String`
- `explorerName - String`

Functionality

Creates an **explorer** with the given **name** of the given **type** and saves it in the repository. If the type is invalid, throw an `IllegalArgumentException` with the following message:

"Explorer type doesn't exists."

Otherwise, the method should **return** the following message:

- **"Added {type}: {explorerName}."**

AddState Command

Parameters

- `stateName - String`
- `exhibits – String... (Varargs)`

Functionality

Creates a **state** with the provided **exhibits** and **name** and save it in the repository.

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

- **"Added state: {stateName}."**

RetireExplorer Command

Parameters

- `explorerName – String`

Functionality

Retires the explorer from Antarctica by removing them from the repository. If an explorer with that name doesn't exist, **throw** `IllegalArgumentException` with the following message:

- **"Explorer {explorerName} doesn't exists."**

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

- **"Explorer {explorerName} has retired!"**

ExploreState Command

Parameters

- `stateName - String`

Functionality

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

- You call each of the explorers and pick only the ones that have energy above 50 units.

- If you **don't have any suitable explorers**, throw an **`IllegalArgumentException`** with the following message: **"You must have at least one explorer to explore the state."**
- After a mission, you must **return the following message** with the **name of the explored state** and the **count** of the **explorers** that **had retired** on the mission:
  "The state {stateName} was explored. {retiredExplorers} researchers have retired on this mission."

FinalResult Command

Functionality

Returns the information about the explorers in the following format:

- If the explorers don't have any suitcase exhibits, print **"None"** on their place.

`"{exploredStatesCount} states were explored."`

`"Information for the explorers:"`

`"Name: {explorerName}"`

`"Energy: {explorerName}"`

`"Suitcase exhibits: {suitcaseExhibits`$_1$`, suitcaseExhibits`$_2$`, suitcaseExhibits`$_3$`, …, suitcaseExhibits `$_n$`}"`

…

`"Name: {explorerName}"`

`"Energy: {explorerName}"`

`"Suitcase exhibits: {suitcaseExhibits`$_1$`, suitcaseExhibits`$_2$`, suitcaseExhibits`$_3$`, …, suitcaseExhibits `$_n$`}"`

Input / Output

You are provided with one interface, which will help you with the correct execution process of your program. The interface is called **Engine** and its **implementational** class should read the input. When the program finishes, the class should print the **output** to the **console**.

Input

These are the input commands:

- **`AddExplorer {explorerType} {explorerName}`**
- **`AddState {stateName}`**
- **`RetireExplorer {explorerName}`**
- **`ExploreState {stateName}`**
- **`FinalResult`**
- **`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
AddExplorer NaturalExplorer John AddExplorer GlacierExplorer Sarah AddExplorer GlacierExplorer Gracia AddExplorer AnimalExplorer Anna AddExplorer NaturalExplorer Michael AddExplorer AnimalExplorer Sam AddExplorer AnimalExplorer Michaela AddExplorer JustExplorer Stam AddState Northern AddState South State RetireExplorer Sarah ExploreState South FinalResult Exit
Output
Added NaturalExplorer: John. Added GlacierExplorer: Sarah. Added GlacierExplorer: Gracia. Added AnimalExplorer: Anna. Added NaturalExplorer: Michael. Added AnimalExplorer: Sam. Added AnimalExplorer: Michaela. Explorer type doesn't exists. Added state: Northern. Added state: South. Explorer Sarah has retired! The state South was explored. 0 researchers have retired on this mission. 1 states were explored. Information for the explorers: Name: John Energy: 53 Suitcase exhibits: State Name: Gracia Energy: 40 Suitcase exhibits: None Name: Anna Energy: 100 Suitcase exhibits: None Name: Michael Energy: 60 Suitcase exhibits: None Name: Sam Energy: 100 Suitcase exhibits: None Name: Michaela Energy: 100 Suitcase exhibits: None

Input

```
AddExplorer GlacierExplorer Jolie
AddState South State
ExploreState South
FinalResult
AddExplorer NaturalExplorer Jack
AddExplorer AnimalExplorer Lia
AddExplorer NaturalExplorer Philip
AddExplorer AnimalExplorer Sarah
AddExplorer AnimalExplorer Thomas
AddState South Northern East West Southwest Southeast
RetireExplorer David
RetireExplorer William
ExploreState South
FinalResult
Exit
```

Output

```
Added GlacierExplorer: Jolie.
Added state: South.
You must have at least one explorer to explore the state.
0 states were explored.
Information for the explorers:
Name: Jolie
Energy: 40
Suitcase exhibits: None
Added NaturalExplorer: Jack.
Added AnimalExplorer: Lia.
Added NaturalExplorer: Philip.
Added AnimalExplorer: Sarah.
Added AnimalExplorer: Thomas.
Added state: South.
Explorer David doesn't exists.
Explorer William doesn't exists.
The state South was explored. 0 researchers have retired on this mission.
1 states were explored.
Information for the explorers:
Name: Jolie
Energy: 40
Suitcase exhibits: None
Name: Jack
Energy: 53
Suitcase exhibits: State
Name: Lia
Energy: 100
Suitcase exhibits: None
Name: Philip
Energy: 60
Suitcase exhibits: None
Name: Sarah
Energy: 100
```

```
Suitcase exhibits: None
Name: Thomas
Energy: 100
Suitcase exhibits: None
```

5. Task 3: Unit Tests (100 points)

You will receive a skeleton with **`Animal`** and **`Farm`** 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!