Exam Preparation – 28 July 2023

Link: https://judge.softuni.org/Contests/3903/Java-OOP-Exam-8-April-2023

1. Overview

We are in the year 2100. Technology is so advanced that robots are all around us. They talk, eat and do whatever you tell them to do.

You are working on a robot service and you need to create a robotService project to monitor the actions of the robot. Each service has a robot that requires different care. Your job is to add, feed and take care of the robot, as well as upgrade it with various supplements.

2. Setup

- Upload **only the robotService** 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.

3. Task 1: Structure (50 points)

You are given interfaces, and you have to implement their functionality in the **correct classes**.

It is not required to implement your structure with **`Engine`**, **`ConsoleReader`**, **`ConsoleWriter`,** and enc. It's good practice but it's not required.

There are **3** types of entities in the application: **`Service`, `Robot`, `Supplement`**.

There should also be **`SupplementRepository`**.

BaseSupplement

`BaseSupplement` is a **base class** of any **type of supplement** and it **should not be able to be instantiated**.

Data

- **`hardness - int`**

- **price - double**
  - The price of the supplements that the service offers.

Constructor

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

```
(int hardness, double price)
```

Child Classes

There are two concrete types of **Supplements**:

PlasticArmor

The plastic armor has a **hardness of 1** and a **price of 10**.

Note: The Constructor **should take no values** upon initialization.

MetalArmor

The metal armor has a **hardness of 5** and a **price of 15**.

Note: The Constructor **should take no values** upon initialization.

BaseRobot

BaseRobot is a **base class** of any **type of robot** 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: **"Robot name cannot be null or empty."**
  - All names are unique.
- **kind - String**
  - If the kind **is null or whitespace,** throw a **NullPointerException** with a message: **"Robot kind cannot be null or empty."**
- **kilograms - int**
  - The kilograms of the **Robot.**
- **price - double**
  - The price of the **Robot.**
  - If the price is below or equal to **0,** throw an **IllegalArgumentException** with a message: **"Robot price cannot be below or equal to 0."**

Behavior

void eating()

The **eating()** method increases the **Robot's** kilograms. Keep in mind that some kinds of **Robot** can implement the method differently.

Constructor

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

```
(String name, String kind, int kilograms, double price)
```

Child Classes

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

FemaleRobot

Has **initial kilograms of 7.**

Can only live in SecondaryService!

The constructor should take the following values upon initialization:

```
(String name, String kind, double price)
```

Behavior

`void eating()`

- The method **increases** the robot's kilograms by **1**.

MaleRobot

Has **initial kilograms of 9.**

Can only live in MainService!

The constructor should take the following values upon initialization:

```
(String name, String kind, double price)
```

Behavior

`void eating()`

- The method **increases** the robot's kilograms by **3**.

BaseService

BaseService is a **base class** of any **type of service** 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: **"Service name cannot be null or empty."**
  - All names are unique.
- **capacity - int**
  - The **number** of **Robot** a **Service can have.**
- **supplements - Collection<Supplement>**
- **robots - Collection<Robot>**

Behavior

`int sumHardness()`

Returns the sum of **each supplement's hardness** in the **Service**.

`void addRobot(Robot robot)`

Adds a **Robot** in the **Service** if there is a **capacity** for it.

If there is **not enough capacity** to **add** the **Robot** in the **Service, throw an IllegalStateException** with **the following message:**

- `"Not enough capacity for this robot."`

`void removeRobot(Robot robot)`

Removes a **Robot** from the **Service**.

`void addSupplement(Supplements supplement)`

Adds a **Supplements** in the **Service**.

`void feeding()`

The **feeding()** method **feeds all robots** in the **Service**.

`String getStatistics()`

Returns a **String** with **information** about the **Service** in the format below.

```
"{serviceName} {serviceType}:
Robots: {robotName1} {robotName2} {robotName3} ... / Robots: none
Supplements: {supplementsCount} Hardness: {sumHardness}"
```

Note: I remind you that there are **two service types – MainService** and **SecondaryService**.

Constructor

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

```
(String name, int capacity)
```

Child Classes

There are 2 concrete types of **Service**:

SecondaryService

Has **15 capacity.**

The constructor should take the following values upon initialization:

```
(String name)
```

MainService

Has **30 capacity.**

The constructor should take the following values upon initialization:

```
(String name)
```

SupplementRepository

The **supplement repository** is a **repository** for the **supplements** that are in the **services**.

Data

- **supplements** - **Collection<Supplement>**

Behavior

`void addSupplement(Supplement supplement)`

- **Adds** a **supplement** to the **collection**.

`boolean removeSupplement(Supplement supplement)`

- **Removes** a **supplement** from the **collection**. **Returns** `true` if the deletion was **successful**, **otherwise** - `false`.

`Supplement findFirst(String type)`

- **Returns** the **first supplement** of the **given type**, if there is any. **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 **`ControllerImpl`** class, which implements the interface and implements all its methods. The given methods should have the following logic:

Data

You need some private fields in your controller class:

- **`supplements`** - **`SupplementRepository`**
- **`services`** - **`Collection<Service>`**

Commands

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

AddService Command

Parameters

- **`type`** - **`String`**
- **`name`** - **`String`**

Functionality

Creates and adds a **`Service`** to the services' collection. **Valid** types are: "**`MainService`**" and "**`SecondaryService`**".

If the **`Service` type** is **invalid**, you have to **throw a `NullPointerException`** with **the following message:**

- **`"Invalid service type."`**

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

- **`"{serviceType} is successfully added."`**

AddSupplement Command

Parameters

- **`type`** - **`String`**

Functionality

Creates a **supplement** of the **given type** and **adds** it to the **SupplementRepository**. **Valid** types are: "**PlasticArmor**" and "**MetalArmor**". If the supplement **type** is **invalid**, throw an **IllegalArgumentException** with a message:

- **"Invalid supplement type."**

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

- **"{supplementType} is successfully added."**

SupplementForService Command

Parameters

- **serviceName - String**
- **supplementType - String**

Functionality

Adds the desired **Supplement** to the **Service** with the **given name**. You have to remove the **Supplement** from the **SupplementRepository** if the insert is **successful**.

If there is **no such supplement**, you have to **throw an IllegalArgumentException** with **the following message**:

- **"Supplement of type {supplementType} is missing."**

If **no exceptions** are **thrown, return** the **String**:

- **"Successfully added {supplementType} to {serviceName}."**

AddRobot Command

Parameters

- **serviceName** - **String**
- **robotType** - **String**
- **robotName** - **String**
- **robotKind** - **String**
- **price** - **double**

Functionality

Creates and adds the desired **Robot** to the **Service** with the **given name**. **Valid Robot** types are: "**MaleRobot**", "**FemaleRobot**".

Note: The method must first check whether the robot type is valid.

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

- **"Invalid robot type."**

If **no errors** are **thrown**, **return** one of the following strings:

- **"Unsuitable service."** - if the given **Robot cannot live** in the **Service.**
  For reference: check their description from **Task 1**.
- **"Successfully added {robotType} to {serviceName}."** - if the **Robot** is **added successfully** in the **Service.**

FeedingRobot Command

Parameters

- **serviceName** - **String**

Functionality

Feeds all **Robot** in the **Service** with the given name.

Returns a **string** with information about **how many robots** were **successfully fed**, in the following **format**:

- **"Robots fed: {fedCount}"**

SumOfAll Command

Parameters

- **serviceName** - **String**

Functionality

Calculates the value of the **Service** with the given name. It is calculated by the sum of all **Robot** and **Supplement** prices in the **Service**.

Return a **string** in the following **format**:

- **"The value of service {serviceName} is {value}."**
  - The **value** should be **formatted** to the **2nd decimal place**!

Statistics Command

Functionality

Returns information about each service. You can use Service's **getStatistics** method to implement the current functionality.

```
"{serviceName} {serviceType}:
Robots: {robotName1} {robotName2} {robotName3} ... / Robots: none
Supplements: {supplementsCount} Hardness: {sumHardness}"
```

```
"{serviceName} {serviceType}:
Robots: {robotName1} {robotName2} {robotName3} ... / Robots: none
Supplements: {supplementsCount} Hardness: {sumHardness}"
...
```

End Command

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.

Input

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

- **AddService {type} {name}**

- `AddSupplement {type}`
- `SupplementForService {serviceName} {supplementType}`
- `AddRobot {serviceName} {robotType} {robotName} {robotKind} {price}`
- `FeedingRobot {serviceName}`
- `SumOfAll {serviceName}`
- `Statistics`
- `End`

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

`AddService SecondaryService ServiceRobotsWorld`
`AddService MainService ServiceTechnicalsWorld`
`AddSupplement PlasticArmor`
`AddSupplement MetalArmor`
`SupplementForService ServiceRobotsWorld PlasticArmor`
`SupplementForService ServiceTechnicalsWorld PlasticArmor`
`SupplementForService ServiceRobotsWorld MetalArmor`
`AddRobot ServiceRobotsWorld FemaleRobot Scrap Robots 321.26`
`AddRobot ServiceRobotsWorld FemaleRobot Sparkles Robots 211.11`
`FeedingRobot ServiceRobotsWorld`
`FeedingRobot ServiceTechnicalsWorld`
`SumOfAll ServiceRobotsWorld`
`FeedingRobot ServiceRobotsWorld`
`Statistics`
`End`

Output

`SecondaryService is successfully added.`
`MainService is successfully added.`
`PlasticArmor is successfully added.`
`MetalArmor is successfully added.`
`Successfully added PlasticArmor to ServiceRobotsWorld.`
`Supplement of type PlasticArmor is missing.`
`Successfully added MetalArmor to ServiceRobotsWorld.`
`Successfully added FemaleRobot to ServiceRobotsWorld.`
`Successfully added FemaleRobot to ServiceRobotsWorld.`
`Robots fed: 2`
`Robots fed: 0`
`The value of service ServiceRobotsWorld is 557.37.`
`Robots fed: 2`
`ServiceRobotsWorld SecondaryService:`
`Robots: Scrap Sparkles`
`Supplements: 2 Hardness: 6`
`ServiceTechnicalsWorld MainService:`
`Robots: none`
`Supplements: 0 Hardness: 0`

Input
`AddService SecondaryService ServiceRobotsWorld`

```
AddRobot ServiceRobotsWorld MaleRobot Scrap Robots 333.47
AddRobot ServiceRobotsWorld FemaleRobot Bruno Robots1 477.40
AddService MainService ServiceRobotsWorldExtend
AddRobot ServiceRobotsWorldExtend FemaleRobot Esmeralda Persian 101.40
AddRobot ServiceRobotsWorldExtend MaleRobot Sputnik Persian1 2542.21
AddRobot ServiceRobotsWorld Invalid Chico Radgol 90.90
AddSupplement MetalArmor
SupplementForService ServiceRobotsWorldExtend MetalArmor
SupplementForService ServiceRobotsWorld MetalArmor
AddSupplement PlasticArmor
SupplementForService ServiceRobotsWorldExtend MetalArmor
FeedingRobot ServiceRobotsWorld
AddRobot ServiceRobotsWorld MaleRobot Invalid Breed -12
Statistics
End
```

Output

```
SecondaryService is successfully added.
Unsuitable service.
Successfully added FemaleRobot to ServiceRobotsWorld.
MainService is successfully added.
Unsuitable service.
Successfully added MaleRobot to ServiceRobotsWorldExtend.
Invalid robot type.
MetalArmor is successfully added.
Successfully added MetalArmor to ServiceRobotsWorldExtend.
Supplement of type MetalArmor is missing.
PlasticArmor is successfully added.
Supplement of type MetalArmor is missing.
Robots fed: 1
Robot price cannot be below or equal to 0.
ServiceRobotsWorld SecondaryService:
Robots: Bruno
Supplements: 0 Hardness: 0
ServiceRobotsWorldExtend MainService:
Robots: Sputnik
Supplements: 1 Hardness: 5
```

Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **Main**, **Robot** and **Service**. **Service** 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 **robots (**with **ServiceTests** inside**)** from the **skeleton**.