# Java OOP Exam – 09 April 2022



**Overview**

*Christmas is over, February 14th too, but the Tooth Fairy is always here. As you know, she can't do it alone, so she has helpers.*

Your task is to create a **Fairy Shop** project, where different types of **Helpers** crafts **Presents**. Each helper has an energy level, which drops while working on a present, and **Instruments** that lose power, again while working on a present.

## Setup

* Upload **only the** fairyShoppackage 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 **returns 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 **4** types of entities in the application: **Helper, Present, Shop, and Instrument**.

There should also have been a **HelperRepository**, as well as **PresentRepository**.

### BaseHelper

BaseHelper is a **base class** or any **type of Helper** 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:

**"Helper name cannot be null or empty."**

* + All names will be unique.
* **energy** – **int**
  + The energy of a helper.
* instruments – Collection<**Instrument>**
  + A collection of a helper's instruments.

#### Constructor

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

(String name, int energy)

#### Behavior

##### void work()

The **work()** method decreases helpers' energy by 10.

* A helper's energy should **not** drop **below** **0** (If the power becomes less than 0, set it to 0).

##### void addInstrument(Instrument instrument)

This method **adds** an **instrument** to the helper's **collection** of instruments.

**boolean canWork()**

This method returns:

* **true** - if the current energy of the helper is **greater** than **0;**
* **false** - otherwise.

#### Child Classes

There are two types of **BaseHelper**:

##### Happy

Initial **energy** units: **100**

The constructorshould take the following values upon initialization:

(String name)

##### Sleepy

Initial **energy** units: **50**

The method work() **decreases** the helpers' energy by additional **5 units**.

The constructorshould take the following values upon initialization:

(String name)

### InstrumentImpl

The **InstrumentImpl** is a class that represents the tool, which a **Helper** uses to craft **Present**.

**It should** be able to be **instantiated**.

#### Data

* **power – int**
  + The power of an instrument.
  + If the **initial** power is below **0,** throw an **IllegalArgumentException** with a message:

**"Cannot create an Instrument with negative power!"**.

#### Constructor

An **InstrumentImpl** should take the following values upon initialization:

(int power)

#### Behavior

##### void use()

The **use()** method **decreases** the instrument's **power** by **10**.

* An instrument's power should **not** drop **below** **0.** (If the power becomes less than 0, set it to 0).

**boolean isBroken()**

This method returns **true** when **power** becomes **equal** to **0.**

### PresentImpl

This is the class that holds information about the **Present** that a **Helper** is working on.

**It should** be able to be **instantiated**.

#### Data

* **name - String** 
  + If the name **is null or whitespace,** throw a **NullPointerException** with a message:

**"Present name cannot be null or empty."**.

* **energyRequired – int** 
  + The energy a present requires in order to be crafted.
  + If the **initial** energy is below **0,** throw an **IllegalArgumentException** with a message:

**"Cannot create a Present requiring negative energy!"**.

#### Constructor

An **PresentImpl** should take the following values upon initialization:

**(String name, int energyRequired)**

#### Behavior

##### void getCrafted()

The **getCrafted()** **decreases** the required energy of the present by **10 units**.

* A present's required energy should **not** drop **below** **0** (If the energy becomes less than 0, set it to 0).

##### boolean isDone()

The **isDone()** method returns **true** if the **energyRequired** reaches **0**.

### ShopImpl

Create a **ShopImpl** class. The **ShopImpl** class holds the main action, which is the **craft** method.

#### Behavior

**void craft(Present present, Helper helper)**

Here is how the **craft** method works:

* The helper starts crafting the present. This is only possible if the helper has energy and an instrument that isn't broken.
* Keep working **until** the present is **done** or the helper has **energy** (and **instruments** to use).
* If at some point the **power** of the current instrument **reaches** or **drops** **below 0**, meaning it is **broken**, then the helper should take the **next instrument** from its collection, if it has **any** **left**.

### HelperRepository

The helper repository is a repository for the helpers working at Fairy’s Shop.

#### Data

* helpers – **a** **collection of helpers**

#### Behavior

##### void add(Helper helper)

* **Adds** a helper to the collection.
* Every helper is **unique** and it is guaranteed that there will not be a helper with the same name.

##### boolean remove(Helper helper)

* **Removes** a helper from the collection.
* Returns **true** if the deletion was **successful.**

##### Helper findByName(String name)

* Returns a **helper** with that **name** if such exists.

##### Collection<Helper> getModels()

* Returns a collection of helpers(**unmodifiable**).

### PresentRepository

The present repository is a repository for presents that await to be crafted.

#### Data

* presents – **a** **collection of presents.**

#### Behavior

##### void add(Present present)

* **Adds** a present to be crafted.
* Every present is **unique** and it is guaranteed that there will not be a present with the same name.

##### boolean remove(Present present)

* **Removes** a present from the collection
* Returns **true** if the deletion was **successful.**

##### Present findByName(String name)

* Returns a **present** with that **name** if such exists.
* It is guaranteed that the present **exists** in the collection.

##### Collection<Present> getModels()

* Returns collection of presents (**unmodifiable**).

## 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 ControllerImplclass, which implements the interface and implements all 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.

#### AddHelper Command

##### Parameters

* **type** – **String**
* **helperName – String**

##### Functionality

Creates a helper with the given name of the given type.

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

**"Helper type doesn't exist!"**

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

**"Successfully added {helperType} named {helperName}!"**

#### AddInstrumentToHelper Command

##### Parameters

* **helperName – String**
* **power – int**

##### Functionality

Creates an instrument with the given power and adds it to the collection of the helper.

If the helper doesn't exist, throw an **IllegalArgumentException** with a message:

**"The helper you want to add an instrument to doesn't exist!"**

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

**"Successfully added instrument with power {instrumentPower} to helper {helperName}!"**

#### AddPresent Command

##### Parameters

* presentName - String
* **energyRequired – int**

##### Functionality

Creates a **present** with the provided **name** and **required energy** and adds it to the corresponding repository.

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

* **"Successfully added Present: {presentName}!"**

#### CraftPresents Command

##### Parameters

* **presentName - String**

##### Functionality

When the craft command is called, the action happens.

You should start crafting the given present, by assigning helpers which are almost ready:

* The helpers that you should select are the ones with energy **above** 50 units.
* The selected helpers start working on the given present one by one until the present is crafted or no more helpers are available. If the present is crafted, you don't need to assign the remaining helpers.
* If no **helpers are ready**, throw **IllegalArgumentException** with the following message:

**"There is no helper ready to start crafting!"**

* After the work is done, you must return the following message, reporting whether the present is done and how many total instruments were broken by the helpers who actually worked on the present:

**"Present {presentName} is {done/not done}. {countBrokenInstruments} instrument/s have been broken while working on it!"**

**Note:** The **name** of the **present** you receive will always be a **valid** one.

#### Report Command

##### Functionality

Returns information about **crafted** **presents** and **helpers**:

**"{countCraftedPresents} presents are done!"**

**"Helpers info:"**

**"Name: {helperName1}"**

**"Energy: {helperEnergy1}"**

**"Instruments: {countInstruments} not broken left"**

**…**

**"Name: {helperNameN}"**

**"Energy: {helperEnergyN}"**

**"Instruments: {countInstruments} not broken left"**

#### Exit 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:

* **AddHelper** **{helperType} {helperName}**
* **AddPresent** **{presentName} {energyRequired}**
* **AddInstrumentToHelper** **{helperName} {power}**
* **CraftPresent {presentName}**
* **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** |
| **AddHelper Sleepy SleepyHead**  **AddHelper Happy Sunshine**  **AddHelper Invalid Sonny**  **AddInstrumentToHelper SleepyHead 10**  **AddInstrumentToHelper Sunshine 20**  **AddInstrumentToHelper Sunshine 20**  **AddInstrumentToHelper Sunshine 30**  **AddInstrumentToHelper Sunshine 10**  **AddInstrumentToHelper Sunshine 30**  **AddInstrumentToHelper Sunshine 20**  **AddInstrumentToHelper Sunshine 40**  **AddPresent Truck 20**  **AddPresent TeddyBear 20**  **AddPresent Doll 50**  **CraftPresent Truck**  **CraftPresent TeddyBear**  **CraftPresent Doll**  **Report**  **Exit** |
| **Output** |
| **Successfully added Sleepy named SleepyHead!**  **Successfully added Happy named Sunshine!**  **Helper type doesn't exist!**  **Successfully added instrument with power 10 to helper SleepyHead!**  **Successfully added instrument with power 20 to helper Sunshine!**  **Successfully added instrument with power 20 to helper Sunshine!**  **Successfully added instrument with power 30 to helper Sunshine!**  **Successfully added instrument with power 10 to helper Sunshine!**  **Successfully added instrument with power 30 to helper Sunshine!**  **Successfully added instrument with power 20 to helper Sunshine!**  **Successfully added instrument with power 40 to helper Sunshine!**  **Successfully added Present: Truck!**  **Successfully added Present: TeddyBear!**  **Successfully added Present: Doll!**  **Present Truck is done. 1 instrument/s have been broken while working on it!**  **Present TeddyBear is done. 2 instrument/s have been broken while working on it!**  **Present Doll is done. 4 instrument/s have been broken while working on it!**  **3 presents are done!**  **Helpers info:**  **Name: SleepyHead**  **Energy: 50**  **Instruments: 1 not broken left**  **Name: Sunshine**  **Energy: 10**  **Instruments: 3 not broken left** |

|  |
| --- |
| **Input** |
| **AddHelper Sleepy Moony**  **AddHelper Sleepy Latey**  **AddHelper Happy Mikey**  **AddHelper Happy Crispy**  **AddInstrumentToHelper Moony 20**  **AddInstrumentToHelper Mikey 180**  **AddInstrumentToHelper Moony 10**  **AddInstrumentToHelper Latey 10**  **AddInstrumentToHelper Crispy 20**  **AddInstrumentToHelper Crispy 10**  **AddInstrumentToHelper Crispy 10**  **AddPresent WoodenTrain 100**  **AddPresent LegoSet 160**  **AddPresent DinosaurPlush 40**  **AddPresent Laptop 500**  **AddPresent Headphones 300**  **CraftPresent WoodenTrain**  **CraftPresent LegoSet**  **CraftPresent DinosaurPlush**  **CraftPresent Laptop**  **CraftPresent Headphones**  **Report**  **Exit** |
| **Output** |
| **Successfully added Sleepy named Moony!**  **Successfully added Sleepy named Latey!**  **Successfully added Happy named Mikey!**  **Successfully added Happy named Crispy!**  **Successfully added instrument with power 20 to helper Moony!**  **Successfully added instrument with power 180 to helper Mikey!**  **Successfully added instrument with power 10 to helper Moony!**  **Successfully added instrument with power 10 to helper Latey!**  **Successfully added instrument with power 20 to helper Crispy!**  **Successfully added instrument with power 10 to helper Crispy!**  **Successfully added instrument with power 10 to helper Crispy!**  **Successfully added Present: WoodenTrain!**  **Successfully added Present: LegoSet!**  **Successfully added Present: DinosaurPlush!**  **Successfully added Present: Laptop!**  **Successfully added Present: Headphones!**  **Present WoodenTrain is done. 0 instrument/s have been broken while working on it!**  **Present LegoSet is not done. 3 instrument/s have been broken while working on it!**  **Present DinosaurPlush is not done. 3 instrument/s have been broken while working on it!**  **Present Laptop is not done. 3 instrument/s have been broken while working on it!**  **Present Headphones is not done. 3 instrument/s have been broken while working on it!**  **1 presents are done!**  **Helpers info:**  **Name: Moony**  **Energy: 50**  **Instruments: 2 not broken left**  **Name: Latey**  **Energy: 50**  **Instruments: 1 not broken left**  **Name: Mikey**  **Energy: 0**  **Instruments: 1 not broken left**  **Name: Crispy**  **Energy: 60**  **Instruments: 0 not broken left** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **Main**, **Gift,** and **GiftFactory**. **GiftFactory** 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 **gifts (**with **GiftFactoryTests** inside**)** from the **skeleton**.