# Java OOP Retake Exam – 19 Dec 2019

1. **Overview**

*Christmas is just around the corner and Santa Claus is getting all the presents ready. But as you know he can't manage on himself, so the dwarfs are helping him and so are you.*

Your task is to create a **Santa Workshop** project, where different types of **Dwarfs** craft **Presents**. Naturally, each dwarf 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** santaWorkshop 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 **4** types of entities in the application: **Dwarf, Present, Workshop, Instrument**.

There should also have be a **DwarfRepository**, as well as **PresentRepository**.

### BaseDwarf

BaseDwarf is a **base class** or any **type of Dwarf** and it **should not be able to be instantiated**.

#### Data

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

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

* + All names will be unique
* **energy** – **int**
  + The energy of a dwarf
  + If the **initial** energy is below **0,** throw an **IllegalArgumentException** with message:

**"Cannot create Dwarf with negative energy!"**

* instruments – Collection<**Instrument>**
  + A collection of a dwarf's instruments

#### Constructor

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

(String name, int energy)

#### Behavior

##### void work()

The **work()** method decreases dwarfs' energy with 10.

* A dwarf'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 dwarf's **collection** of instruments.

**boolean canWork()**

This method returns:

* **true**, if the current energy of the dwarf is **greater** than **0**
* **false**, otherwise

#### Child Classes

There are two types of **BaseDwarf**:

##### Happy

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

Constructorshould take the following values upon initialization:

(String name)

##### Sleepy

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

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

Constructorshould take the following values upon initialization:

(String name)

### InstrumentImpl

The **InstrumentImpl** is a class that represents the tool, which a **Dwarf** 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 message:

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

#### Constructor

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

(int power)

#### Behavior

##### void use()

The **use()** method **decreases** instrument's **power** with **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 which holds information about the **Present** that a **Dwarf** is working on.

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

#### Data

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

**"Cannot create 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**.

##### boolean isDone()

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

### WorkshopImpl

The **WorkshopImpl** class holds the main action, which is the **craft** method.

#### Behavior

**void craft(Present present, Dwarf dwarf)**

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

* The dwarf starts crafting the present. This is only possible, if the dwarf has energy and an instrument that isn't broken.
* Keep working **until** the present is **done** or the dwarf 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 dwarf should take the **next instrument** from its collection, if it has **any** **left**.

### DwarfRepository

The dwarf repository is a repository for the dwarfs working at Santa's Workshop.

#### Data

* dwarfs – **a** **collection of dwarfs**

#### Behavior

##### void add(Dwarf dwarf)

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

##### boolean remove(Dwarf dwarf)

* **Removes** a dwarf from the collection
* Returns **true** if the deletion was **sucessful**

##### Dwarf findByName(String name)

* Returns a **dwarf** with that **name** if such exists

##### Collection<Dwarf> getModels()

* Returns a collection of dwarfs (**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 **sucessful**

##### 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.

#### AddDwarf Command

##### Parameters

* **type** – **String**
* **dwarfName – String**

##### Functionality

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

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

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

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

* **"Successfully added {dwarfType} named {dwarfName}!"**

#### AddInstrumentToDwarf Command

##### Parameters

* **dwarfName – String**
* **power – int**

##### Functionality

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

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

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

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

**"Successfully added instrument with power {instrumentPower} to dwarf {dwarfName}!"**

#### AddPresent Command

##### Parameters

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

##### Functionality

Creates a **present** with the provided **name** and **required energy**.

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 dwarfs which are most ready:

* The dwarfs that you should select are the ones with energy **above** 50 units.
* The suitable ones start working on the given present.
* If no **dwarfs are ready**, throw **IllegalArgumentException** with the following message:

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

* After the work is done, you must return the following message, reporting whether the present is done and how many instruments have been broken in the process:

**"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 **dwarfs**:

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

**"Dwarfs info:"**

**"Name: {dwarfName1}"**

**"Energy: {dwarfEnergy1}"**

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

**…**

**"Name: {dwarfNameN}"**

**"Energy: {dwarfEnergyN}"**

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

### 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:

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

* **AddDwarf** **{dwarfType} {dwarfName}**
* **AddPresent** **{presentName} {energyRequired}**
* **AddInstrumentToDwarf** **{dwarfName}**
* **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** |
| **AddDwarf Sleepy SleepyHead**  **AddDwarf Happy Sunshine**  **AddDwarf Invalid Sonny**  **AddInstrumentToDwarf SleepyHead 10**  **AddInstrumentToDwarf Sunshine 20**  **AddInstrumentToDwarf Sunshine 20**  **AddInstrumentToDwarf Sunshine 30**  **AddInstrumentToDwarf Sunshine 10**  **AddInstrumentToDwarf Sunshine 30**  **AddInstrumentToDwarf Sunshine 20**  **AddInstrumentToDwarf 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.**  **Dwarf type doesn't exist!**  **Successfully added instrument with power 10 to dwarf SleepyHead!**  **Successfully added instrument with power 20 to dwarf Sunshine!**  **Successfully added instrument with power 20 to dwarf Sunshine!**  **Successfully added instrument with power 30 to dwarf Sunshine!**  **Successfully added instrument with power 10 to dwarf Sunshine!**  **Successfully added instrument with power 30 to dwarf Sunshine!**  **Successfully added instrument with power 20 to dwarf Sunshine!**  **Successfully added instrument with power 40 to dwarf 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!**  **Dwarfs info:**  **Name: SleepyHead**  **Energy: 50**  **Instruments: 1 not broken left**  **Name: Sunshine**  **Energy: 10**  **Instruments: 3 not broken left** |

|  |
| --- |
| **Input** |
| **AddDwarf Sleepy Moony**  **AddDwarf Sleepy Latey**  **AddDwarf Happy Mikey**  **AddDwarf Happy Crispy**  **AddInstrumentToDwarf Moony 20**  **AddInstrumentToDwarf Mikey 180**  **AddInstrumentToDwarf Moony 10**  **AddInstrumentToDwarf Latey 10**  **AddInstrumentToDwarf Crispy 20**  **AddInstrumentToDwarf Crispy 10**  **AddInstrumentToDwarf 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 dwarf Moony!**  **Successfully added instrument with power 180 to dwarf Mikey!**  **Successfully added instrument with power 10 to dwarf Moony!**  **Successfully added instrument with power 10 to dwarf Latey!**  **Successfully added instrument with power 20 to dwarf Crispy!**  **Successfully added instrument with power 10 to dwarf Crispy!**  **Successfully added instrument with power 10 to dwarf 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!**  **Dwarfs 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 **Present** and **PresentBag** 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**.

Note: The PresentBag you need to test is in the **package** christmas, so zip package christmas.

Do **NOT** use **Mocking** in your unit tests!