## C# OOP Retake Exam - 19 December 2019

## 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** SantaWorkshopproject in every problem **except** **Unit Tests**
* **Do not modify the interfaces or their namespaces**
* 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** or **properties** in the concrete class than the interface has defined
* Make sure you have **no public fields** anywhere

## Task 1: Structure (50 points)

For this task’s evaluation logic in the methods isn’t included.

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 and Instrument**. There should also be **DwarfRepository and PresentRepository**.

### Dwarf

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

#### Data

* **Name** - **string**
  + ~~If the name~~ **~~is null or whitespace,~~** ~~throw an~~ **~~ArgumentException~~** ~~with message:~~

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

* + All names will be unique
* **Energy** - **int**
  + ~~The energy of a dwarf~~
  + ~~If a dwarf’s~~ **~~energy drops below 0~~**~~,~~ **~~set it to 0~~**
* Instruments - ICollection<I**Instrument>**
  + A collection of a dwarf's instruments

#### Constructor

~~A~~ **~~Dwarf~~** ~~should take the following values upon initialization:~~

~~string name, int energy~~

#### Behavior

##### abstract void Work()

~~The~~ **~~Work()~~** ~~method decreases the dwarf's energy by 10.~~

* ~~If a dwarf’s energy~~ **~~drops below 0~~**~~,~~ **~~set it to 0~~**~~.~~

##### void AddInstrument(IInstrument instrument)

~~This method~~ **~~adds~~** ~~the given~~ **~~instrument~~** ~~to the dwarf's~~ **~~collection~~** ~~of instruments.~~

#### Child Classes

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

##### HappyDwarf

**~~Has 100 initial energy.~~**

~~A~~ **~~HappyDwarf~~** ~~should take the following values upon initialization:~~

~~string name~~

##### SleepyDwarf

**~~Has 50 initial energy.~~**

~~A~~ **~~SleepyDwarf~~** ~~should take the following values upon initialization:~~

~~string name~~

###### **~~Behavior~~**

~~The method Work()~~ **~~decreases~~** ~~the dwarf's energy by additional~~ **~~5 units~~** ~~(15 in total).~~

### Instrument

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

#### Data

* **Power** - **int**
  + ~~The power of an instrument~~
  + ~~If the power is below~~ **~~0,~~****~~set it to 0~~**~~.~~

#### 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~~

**bool IsBroken()**

* ~~This method returns~~ **~~true~~** ~~if the~~ **~~power~~** ~~is~~ **~~equal~~** ~~to~~ **~~0~~**

#### Constructor

~~An~~ **~~Instrument~~** ~~should take the following values upon initialization:~~

~~int power~~

### Present

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

#### Data

* **Name** - **string**
  + ~~The name of a present~~
  + ~~If the name~~ **~~is null or whitespace,~~** ~~throw an~~ **~~ArgumentException~~** ~~with message:~~

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

* **EnergyRequired** - **int**
  + ~~The energy a present requires in order to be crafted~~
  + ~~If the~~ **~~energyRequired~~** ~~is~~ **~~below 0~~**~~,~~ **~~set it to 0~~**

#### Behavior

##### void GetCrafted()

~~The~~ **~~GetCrafted()~~** ~~method~~ **~~decreases~~** ~~the required energy of the present by~~ **~~10 units~~**~~.~~

* ~~A present's required energy should~~ **~~not~~** ~~drop~~ **~~below~~****~~0~~**~~.~~

##### bool IsDone()

~~The~~ **~~IsDone()~~** ~~method returns~~ **~~true~~** ~~if the~~ **~~energyRequired~~** ~~is equal to~~ **~~0~~**~~.~~

#### Constructor

~~A~~ **~~Present~~** ~~should take the following values upon initialization:~~

~~string name, int energyRequired~~

### Workshop

The **Workshop** class holds the main action, which is the **Craft** method.

#### Behavior

**void Craft(IPresent present, IDwarf 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.
* At the same time the present is getting crafted, so call the **GetCrafted()** method for the present.
* 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**.

#### Constructor

~~A~~ **~~Workshop~~** ~~should take no values upon initialization.~~

### DwarfRepository

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

#### Data

* ~~Models -~~ **~~a~~****~~collection of dwarfs (unmodifiable)~~**

#### Behavior

##### void Add(IDwarf dwarf)

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

**bool Remove(IDwarf dwarf)**

* **~~Removes~~** ~~a~~ **~~dwarf~~** ~~from the~~ **~~collection~~**~~.~~ **~~Returns true~~** ~~if the deletion was~~ **~~sucessful~~**~~,~~ **~~otherwise~~** ~~-~~ **~~false~~**~~.~~

**IDwarf FindByName(string name)**

* **~~Returns~~** ~~the~~ **~~first~~****~~dwarf~~** ~~with the~~ **~~given name~~**~~, if such exists.~~ **~~Otherwise~~**~~, returns~~ **~~null~~**~~.~~

### PresentRepository

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

#### Data

* ~~Models -~~ **~~a~~****~~collection of presents (unmodifiable)~~**

#### Behavior

##### void Add(IPresent present)

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

**bool Remove(IPresent present)**

* **~~Removes~~** ~~a~~ **~~present~~** ~~from the~~ **~~collection~~**~~.~~ **~~Returns true~~** ~~if the deletion was~~ **~~sucessful~~**~~,~~ **~~otherwise~~** ~~-~~ **~~false~~**~~.~~

**IPresent FindByName(string name)**

* **~~Returns~~** ~~the~~ **~~first~~****~~present~~** ~~with the~~ **~~given name~~**~~, if such exists.~~ **~~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, which you have to implement in the correct classes.

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

The first interface is **I**Controller. You must create a Controllerclass, which implements the interface and implements all of its methods. The constructor of Controllerdoes not take any arguments. The given methods should have the logic described for each in the Commands section.

### Data

~~You need to keep track of some things, this is why you need some private fields in your controller class:~~

* **~~dwarfs~~** ~~-~~ **~~DwarfRepository~~**
* **~~presents~~** ~~-~~ **~~PresentRepository~~**

### Commands

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

#### AddDwarf Command

##### Parameters

* dwarfType - string
* dwarfName - string

##### Functionality

**~~Adds~~** ~~a Dwarf.~~ **~~Valid~~** ~~types are: "~~**~~HappyDwarf~~**~~" and "~~**~~SleepyDwarf~~**~~".~~

~~If the~~ **~~Dwarf~~****~~type~~** ~~is~~ **~~invalid~~**~~, you have to~~ **~~throw an InvalidOperationException~~** ~~with~~ **~~the following message:~~**

* **~~"Invalid dwarf type."~~**

~~If the~~ **~~Dwarf~~** ~~is~~ **~~added successfully~~**~~, the method should~~ **~~return~~** ~~the following~~ **~~string~~**~~:~~

* **~~"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~~ **~~InvalidOperationException~~** ~~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}!"~~**

#### CraftPresent 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 (first the dwarfs with the most energy):

* The dwarfs that you should select are the ones with energy **equal to or above 50 units**.
* The **suitable ones** start working on the given present.
* If a dwarf’s **energy becomes 0**, **remove it from the repository**.
* If no **dwarfs are ready**, throw **InvalidOperationException** 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:

**"Present {presentName} is {done/not done}."**

**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"**

**Note: Use \r\n or Environment.NewLine for a new line.**

#### Exit Command

##### Functionality

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 IEngine 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:

* **AddDwarf** **{dwarfType} {dwarfName}**
* **AddPresent** **{presentName} {energyRequired}**
* **AddInstrumentToDwarf** **{dwarfName} {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** |
| **AddDwarf SleepyDwarf SleepyHead**  **AddDwarf HappyDwarf Sunshine**  **AddDwarf InvalidDwarf 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 SleepyDwarf named SleepyHead.**  **Successfully added HappyDwarf named Sunshine.**  **Invalid dwarf type**  **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.**  **Present TeddyBear is done.**  **Present Doll is done.**  **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 SleepyDwarf Moony**  **AddDwarf SleepyDwarf Latey**  **AddDwarf HappyDwarf Mikey**  **AddDwarf HappyDwarf 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 SleepyDwarf named Moony.**  **Successfully added SleepyDwarf named Latey.**  **Successfully added HappyDwarf named Mikey.**  **Successfully added HappyDwarf 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.**  **Present LegoSet is not done.**  **Present DinosaurPlush is not done.**  **Present Laptop is not done.**  **Present Headphones is not done.**  **1 presents are done!**  **Dwarfs info:**  **Name: Moony**  **Energy: 5**  **Instruments: 0 not broken left**  **Name: Latey**  **Energy: 35**  **Instruments: 0 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 **Bag** 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!