C# OOP Exam – 8 April 2023

RobotService

1. Overview

We are in the year 2100. Technology is so advanced that robots are all around us. They are autonomous and do whatever you tell them to do. They use fluidization instead of charging to provide the energy they need, so robots need to be fed.

You are working on a **robot service** and you need to create a **RobotService** project to monitor the actions of a 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** project 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.
- **Exception messages** and **output messages** can be found in the **`"Utilities"`** folder.
- For solving this problem use **Visual Studio 2019, Visual Studio 2022** and **netcoreapp 3.1, netcoreapp 6.0**

3. Task 1: Structure (50 points)

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

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

There are **2** types of entities in the application: **`Supplement`** and **`Robot`**.

There should also be **`SupplementRepository`** and **`RobotRepository`**, both implementing the **`IRepository`** interface.

Supplement

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

Data

- **`InterfaceStandard – int`**
  - The compatibility standard that the Supplement supports.
- **`BatteryUsage - int`**
  - The power that the **`Supplement`** will consume additionally when installed to a **`Robot`**.

Constructor

A **`Supplement`** should take the following values upon initialization:

```
int interfaceStandard, int batteryUsage
```

Child Classes

There are two concrete types of **`Supplement`**:

SpecializedArm

A **`SpecializedArm`** has an **`InterfaceStandard`** of **10045** and a **`BatteryUsage`** of **10 000** mAh.

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

LaserRadar

A **`LaserRadar`** has an **`InterfaceStandard`** of **20082** and a **`BatteryUsage`** of **5 000** mAh.

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

Robot

A **`Robot`** is a **base class** of any **type of robot** and it **should not be able to be instantiated**.

Data

- **`Model – string`**
  - If the **`Model`** is **null or whitespace,** throw a new **`ArgumentException`** with the message: `"Model cannot be null or empty."`
- **`BatteryCapacity - int`**
  - The maximum charging level of the **`Robot`** battery.
  - The **`BatteryCapacity`** cannot drop below zero. If it does, throw a new **`ArgumentException`** with the message: `"Battery capacity cannot drop below zero."`
- **`BatteryLevel – int`**
  - The current level of the battery. When creating a **new `Robot`**, set its initial value, equal to the **`BatteryCapacity`**.
- **`ConvertionCapacityIndex - int`**
  - The ability of the **`Robot`** to convert food into energy.
- **`InterfaceStandards – IReadOnlyCollection<int>`**
  - A collection of all the supported connectivity standards by a specific **`Robot.`**

Behavior

void Eating(int minutes)

The **Robot** will be in fluidization mode, so it will convert the food into electrical energy. For **every minute of eating**, it will **produce energy** equal to the **ConvertionCapacityIndex multiplied by the given minutes.**

- The **Eating()** method increases the **Robot's BatteryLevel,** with the produced energy.
- If the battery is **fully charged (BatteryLevel = BatteryCapacity)**, the **eating stops earlier**.

void InstallSupplement(ISupplement supplement)

- The **InstallSupplemet()** method takes the given supplement's **InterfaceStandard** and adds it to the list of **InterfaceStandards** of the **Robot.**
- Decreases the **BatteryCapacity** of the robot by the **BatteryUsage** of the supplement.
- Decreases the **BatteryLevel** of the robot by the **BatteryUsage** of the supplement.

bool ExecuteService(int consumedEnergy)

The **ExecuteService()** method decreases the **Robot's BatteryLevel,** with the given amount of consumed energy.

- If the **BatteryLevel** is **equal or greater than** the given **consumedEnergy**, decrease the **BatteryLevel** with the given amount of **consumedEnergy** and return **True**.
- If the **BatteryLevel** is **less than** the given **consumedEnergy**, it means that it is **NOT enough**. **Skip the execution** and return **False**.

Override ToString() method:

Override the existing method **ToString()** and modify it, so the returned string must be in the following format:

```
"{robotTypeName} {Model}:
--Maximum battery capacity: {BatteryCapacity}
--Current battery level: {BatteryLevel}
--Supplements installed: {standard1} {standard2}…/none"
```

Note: For best clarity see the output examples!

Constructor

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

```
string model, int batteryCapacity, int conversionCapacityIndex
```

Child Classes

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

DomesticAssistant

Has **BatteryCapacity of 20 000 mAh.**

The **DomesticAssistant** will produce a capacity of 2000 mAh of energy for every minute of eating - **(convertionCapacityIndex = 2 000).**

The Constructor of the **DomesticAssistant** should take the following parameters upon initialization:

```
string model
```

IndustrialAssistant

Has **BatteryCapacity of 40 000 mAh.**

The **IndustrialAssistant** will produce a capacity of 5000 mAh of energy for every minute of eating - **(convertionCapacityIndex = 5 000).**

The Constructor of the **IndustrialAssistant** should take the following parameters upon initialization:

```
string model
```

SupplementRepository

The **SupplementRepository** is an **IRepository<ISupplement>. Collection** for the **supplements** that are created in the application.

Data

- **A private field would be useful to store the items added.**

Behavior

```
IReadOnlyCollection<ISupplement> Models()
```

- **Returns** all added items as a readonly collection.

```
void AddNew(ISupplement supplement)
```

- **Adds** a new **ISupplement** to the **SupplementRepository**.

```
bool RemoveByName(string typeName)
```

- **Removes** the first **ISupplement** from the **collection,** which has the same typeName as the given **typeName**. **Returns true** if the removal was **successful**, **otherwise** returns **false**.

```
ISupplement FindByStandard(int interfaceStandard)
```

- Returns the **first ISupplement supporting the given interface**, if there is any. Otherwise, returns **null**.

RobotRepository

The **RobotRepository** is an **IRepository<IRobot>. Collection** for the **robots** that are created in the application.

Data

- **A private field would be useful to store the items added.**

Behavior

```
IReadOnlyCollection<IRobot> Models()
```

- **Returns** all added items as a readonly collection.

```
void AddNew(IRobot robot)
```

- **Adds** a new **IRobot** to the **RobotRepository**.

```
bool RemoveByName(string robotModel)
```

- **Removes** the first **IRobot** from the **collection,** which **Model** is the same as the given **robotModel**. **Returns true** if the deletion was **successful**, **otherwise** returns **false**.

IRobot FindByStandard(int interfaceStandard)

- Returns the **first IRobot supporting the given interface**, 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**, which you have to implement in the correct class.

The interface is **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of the **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the **Controller** class, go into the **Engine** class constructor and uncomment the "`this.controller = new Controller();`" line.

Data

You will need some private fields in your controller class:

- **supplements** - **SupplementRepository**
- **robots** - **RobotRepository**

Commands

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

CreateRobot Command

Parameters

- **model** - **string**
- **typeName** - **string**

Functionality

The method should **create and add** a new **IRobot** to the **RobotRepository.**

- If the given **typeName** is **NOT** presented as a valid Robot's child class (**DomesticAssistant or IndustrialAssistant**), return the following message: "`Robot type {typeName} cannot be created.`"
- If the above case is **NOT** reached, **create an IRobot from the valid child type** and **add** it to the **RobotRepository**. Return the following message: "`{typeName} {model} is created and added to the RobotRepository.`"

CreateSupplement Command

Parameters

- **typeName** - **string**

Functionality

The method should **create and add** a new **ISupplement** to the **SupplementRepository.**

- If the given **typeName** is **NOT** presented as a valid Supplement's child class (**SpecializedArm or LaserRadar**), return the following message: "`{typeName} is not compatible with our robots.`"

- If **the above case** is NOT reached, **create** a new **ISupplement** and **add** it to the **SupplementRepository**. Return the following message: **"{typeName} is created and added to the SupplementRepository."**

UpgradeRobot Command

Parameters

- **model** - **string**
- **supplementTypeName** - **string**

Functionality

This method will upgrade a robot with a new supplement. There will always be **at least one supplement from the correct type** already added to the **SupplementRepository**. There will always be **at least one robot from the given model** already added to the **RobotRepository**:

1. Find the **first ISupplement** with the given **supplementTypeName** in the **SupplementRepository** and take its **interface value**.
2. From the **RobotRepository**, take only the robots, **NOT** supporting the **interface value** (***check if every robot's InterfaceStandards collection NOT containing the* interface value**).
3. **Select only the robots**, from the given model (***check if every robot's Model is equal to the given model***).
4. If the **collection is empty**, that means all of the robots in the **RobotRepository** from the given **model**, are already upgraded with a **Supplement** from the given **supplementTypeName**,
    - return the following message: **"All {model} are already upgraded!"**
5. If there are still not upgraded robots, take the first **IRobot** from the previous selected robots and use the built-in **InstallSupplement()** method to upgrade the robot with the new supplement.
    - Remove the **ISupplement** from the **SupplementRepository**.
    - Return the following message: **"{model} is upgraded with {supplementTypeName}."**

PerformService Command

Parameters

- **serviceName** – **string**
- **interfaceStandard - int**
- **totalPowerNeeded - int**

Functionality

To perform a specific service, you will need **only** robots supporting the given **interfaceStandard**. You will have to check the **InterfaceStandarts** property of **every single robot** from the **RobotRepository** and take those which **meet that requirement**.

1. **Select the robots**, supporting the given **interfaceStandard** from the **RobotRepository** (***check if every robot's InterfaceStandards collection contains the given interfaceStandard***)
2. If **NONE** of the robots in the **RobotRepository** supports the given **interfaceStandard**, return the following message: **"Unable to perform service, {intefaceStandard} not supported!"**
3. **Order** the selected robots **by BatteryLevel descending**.
4. Find the **sum of the BatteryLevel** of **the selected robots**.

5. **If the sum** of the **`BatteryLevel`** of the **selected robots**, **is less than** the **`totalPowerNeeded`**,
    - Return the following message:

`"{serviceName} cannot be executed! {totalPowerNeeded - availablePower} more power needed."`

6. Else if the **`totalPowerNeeded`**, **is greater or equal** to the **`BatteryLevel sum`**, **each of the selected robots** will work on the service **until the service is performed successfully (`totalPowerNeeded == 0`)**:
    - **Create a counter** to calculate **how many robots will take part** in the service.
    - If **`robot.BatteryLevel >= totalPowerNeeded`**
        - Extract energy from the battery, equal to the **`totalPowerNeeded`** (HINT: *robot.ExecuteService(totalPowerNeeded)*)
        - **Increase the counter by 1** and **stop executing the service**.
    - If **`robot.BatteryLevel < totalPowerNeeded:`**
        - Decrease the **`totalPowerNeeded`** with the value of **`robot.BatteryLevel`**
        - **Extract all the energy from the battery** (HINT: *robot.ExecuteService(robot.BatteryLevel)*)
        - **Increase the counter by 1 and proceed with the next robot.**
7. When the service is performed successfully, return the following message: **`"{serviceName} is performed successfully with {usedRobotsCount} robots."`**

RobotRecovery Command

Parameters

- **`model – string`**
- **`minutes - int`**

Functionality

Feed all robots in the **`RobotRepository`** from the given **`model`** for the given count of **`minutes`**. Choose only those robots that have **`BatteryLevel`** **under 50%** from the total **`BatteryCapacity.`**

Remember that when feeding a robot, it will be in **fluidization mode** and it will **convert food into energy**. That means that after feeding, the robot's **`BatteryLevel`** should be **increased**. Use the built-in **`Eating()`** method of each robot.

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

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

Report Command

Functionality

Returns information about each robot from the **`RobotRepository`**. Arrange the robots by **`BatteryLevel`**, **descending**, then by **`BatteryCapacity`**, **ascending**. In order to receive correct output, use the **`ToString()`** method **of each robot:**

`"{robot1}`

`{robot2}`

`...`

`{robotn}"`

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:

- **`CreateRobot {model} {typeName}`**
- **`CreateSupplement {typeName}`**
- **`UpgardeRobot {model} {supplementTypeName}`**
- **`PerformService {serviceName} {interfaceStandard} {totalPowerNeeded}`**
- **`RobotRecovery {model} {minutes}`**
- **`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

`CreateRobot K-2SO IndustrialAssistant`
`CreateRobot T-X IndustrialAssistant`
`CreateRobot AVA DomesticAssistant`
`CreateRobot KUSANAGI IndustrialAssistant`
`CreateRobot C-3PO DomesticAssistant`
`CreateRobot R2-D2 DomesticAssistant`
`CreateRobot C1-10P SocialAssistant`
`CreateRobot C-3PO DomesticAssistant`
`CreateSupplement FaceRecognitionCamera`
`CreateSupplement SpecializedArm`
`CreateSupplement SpecializedArm`
`CreateSupplement SpecializedArm`
`CreateSupplement SpecializedArm`
`CreateSupplement LaserRadar`
`CreateSupplement LaserRadar`
`CreateSupplement LaserRadar`
`CreateSupplement LaserRadar`
`PerformService Dishwashing 10045 1000`
`UpgradeRobot C-3PO SpecializedArm`
`UpgradeRobot C-3PO SpecializedArm`
`UpgradeRobot C-3PO SpecializedArm`
`UpgradeRobot C-3PO LaserRadar`
`UpgradeRobot R2-D2 SpecializedArm`
`UpgradeRobot KUSANAGI LaserRadar`
`UpgradeRobot KUSANAGI SpecializedArm`
`PerformService PaintRoad 20082 100000`
`PerformService DishWashing 10045 1000`
`PerformService AutomotiveAssembly 10045 25000`
`RobotRecovery C-3PO 3`
`RobotRecovery KUSANAGI 3`
`Report`

```
Exit
```

Output

```
IndustrialAssistant K-2SO is created and added to the RobotRepository.
IndustrialAssistant T-X is created and added to the RobotRepository.
DomesticAssistant AVA is created and added to the RobotRepository.
IndustrialAssistant KUSANAGI is created and added to the RobotRepository.
DomesticAssistant C-3PO is created and added to the RobotRepository.
DomesticAssistant R2-D2 is created and added to the RobotRepository.
Robot type SocialAssistant cannot be created.
DomesticAssistant C-3PO is created and added to the RobotRepository.
FaceRecognitionCamera is not compatible with our robots.
SpecializedArm is created and added to the SupplementRepository.
SpecializedArm is created and added to the SupplementRepository.
SpecializedArm is created and added to the SupplementRepository.
SpecializedArm is created and added to the SupplementRepository.
LaserRadar is created and added to the SupplementRepository.
LaserRadar is created and added to the SupplementRepository.
LaserRadar is created and added to the SupplementRepository.
LaserRadar is created and added to the SupplementRepository.
Unable to perform service, 10045 not supported!
C-3PO is upgraded with SpecializedArm.
C-3PO is upgraded with SpecializedArm.
All C-3PO are already upgraded!
C-3PO is upgraded with LaserRadar.
R2-D2 is upgraded with SpecializedArm.
KUSANAGI is upgraded with LaserRadar.
KUSANAGI is upgraded with SpecializedArm.
PaintRoad cannot be executed! 70000 more power needed.
DishWashing is performed successfully with 1 robots.
AutomotiveAssembly is performed successfully with 2 robots.
Robots fed: 0
Robots fed: 1
IndustrialAssistant K-2SO:
--Maximum battery capacity: 40000
--Current battery level: 40000
--Supplements installed: none
IndustrialAssistant T-X:
--Maximum battery capacity: 40000
--Current battery level: 40000
--Supplements installed: none
DomesticAssistant AVA:
--Maximum battery capacity: 20000
--Current battery level: 20000
--Supplements installed: none
IndustrialAssistant KUSANAGI:
--Maximum battery capacity: 25000
--Current battery level: 15000
--Supplements installed: 20082 10045
DomesticAssistant C-3PO:
--Maximum battery capacity: 10000
--Current battery level: 10000
--Supplements installed: 10045
DomesticAssistant R2-D2:
--Maximum battery capacity: 10000
--Current battery level: 9000
--Supplements installed: 10045
DomesticAssistant C-3PO:
--Maximum battery capacity: 5000
--Current battery level: 5000
```

```
--Supplements installed: 10045 20082
```

Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **`Factory`**, **`Robot`** and **`Supplement`**. **`Factory`** 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. If some of the methods in **`Factory`** **change anything from the other classes, you should cover that functionality also.** In Judge, you upload **.zip (**with **`RobotFactory.Tests`** inside**)** from the **skeleton**